

For Reference

NOT TO BE TAKEN FROM THIS ROOM

Ex LIBRIS
UNIVERSITATIS
ALBERTAEENSIS



THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR Johannes Siegfried Weiss
TITLE OF THESIS A Data Acquisition and Transmission
 System
DEGREE FOR WHICH THESIS WAS PRESENTED Master of Science
YEAR THIS DEGREE GRANTED FALL 1982

Permission is hereby granted to THE UNIVERSITY OF ALBERTA LIBRARY to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

THE UNIVERSITY OF ALBERTA

A Data Acquisition and Transmission System

by



Johannes Siegfried Weiss

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE
OF Master of Science

Department of Electrical Engineering

The University of Alberta

FALL 1982

THE UNIVERSITY OF ALBERTA
FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled A Data Acquisition and Transmission System submitted by J. S. Weiss in partial fulfilment of the requirements for the degree of Master of Science.

To my parents, Margit and Siegfried Weiss

Abstract

The requirements, design, and performance of a microprocessor based data acquisition and transmission system are described. This system is to sample and digitize a number of low frequency analog signals and to transmit this information to a time shared main frame computer. Features of the data acquisition process include the concurrent sampling of 8 signals, each at a rate of 40 hertz, an 8 channel digital filter, and a selectable resampling rate. Features of the transmission process include a large nonvolatile data buffer, in the form of a bubble memory, and a handshaking mechanism whereby data can be reliably sent to the main frame computer. Concurrent data acquisition and transmission minimizes the time between data acquisition and data analysis. The system is intended to aid biomedical research. Consequently, there is an emphasis on reliability and convenient operation.

Acknowledgments

I wish to thank my mother, Margit Weiss and my brother, Michael Weiss, for proof reading and discussing this thesis. I wish to thank my supervising professor, Y.J. Kingma, for providing suggestions on how to improve this thesis and the design of the project it describes.

Funding has been provided by NSERC.

Table of Contents

Chapter	Page
I. Motivation	1
II. Requirements	4
A. Microcomputer	6
B. Data Acquisition Process	6
C. Transmission Process	8
D. Data Buffer	9
E. Control	11
F. Other Devices	11
III. Specifications	13
A. Microcomputer	13
B. Data Acquisition Process	13
Initial Sampling	14
Digital Filter	14
Resampling	15
C. Transmission Process	15
D. Data Buffer	18
E. Control	20
F. Other Devices	21
IV. Implementation	22
A. Microcomputer System	22
B. Data Acquisition Process	25
Initial Sampling	25
Digital Filter	27
Resampling	30
C. Transmission Process	31

Transmission Interface Routines	31
Transmission Process Driver	35
Data Format	35
D. Bubble Memory System	39
E. Control	46
F. Other Devices	51
V. Performance	54
A. Microcomputer System	54
B. Data Acquisition Process	55
Initial Sampling	55
Digital Filter	56
C. Transmission Process	57
D. Bubble Memory System	59
E. Control	60
F. Other Devices	60
G. System Timing	60
VI. Applications	67
VII. Refinements and Improvements	71
Bibliography	74
Appendix A -- Operator's Manual	76
General Function	76
Function Keys	78
Typical Operation	86
Special Modes of Operation	88
Problems	89
Appendix B -- Microcomputer Software Listing	91
ROM1	91

ROM2	109
ROM3	126
ROM4	144
ROM5	162
ROM6	178
ROM7	210
ROM8	241
Appendix C -- Fortran Support	273
SPRNTD	273
SPROCESS	283
SDFIL	284
Appendix D -- Schematic Diagrams	289

List of Tables

Table	Page
IV.1 The Sets of Filter Coefficients	30
V.1 Routine Execution Times	61

List of Figures

Figure	Page
II.1 Environment of the System	5
II.2 The Data Acquisition and Transmission System	5
III.1 Connection to the MTS System	16
IV.1 The Microcomputer System	24
IV.2 Implementation of Initial Sampling	26
IV.3 Digital Filter Implementation	28
IV.4 Structure of the Transmission Interface Routines	32
IV.5 The Transmission Process Driver	36
IV.6 Example of the Data Format	38
IV.7 Print Out of the Example Data Format	39
IV.8 Structure of the Bubble Memory System Software	42
IV.9 System Control State Diagram	47
V.1 Read and Write Page Timing	63
V.2 System Timing	64
VI.1 Analysis of Electrical Activity	68
VI.2 Analysis of Changes in Displacement	68
VI.3 Analysis of Triangular Wave	69

I. Motivation

The development of a data acquisition and transmission system has been motivated by a need to record and analyse activity within the gastrointestinal tract or within tissues taken from that tract. By closely observing the different types of activity and the conditions under which that activity occurs, insight into the mechanisms and functions of the gastrointestinal tract may be obtained. Some of the activities that are being studied are variations in electrical potentials, pressures, displacements, and forces.

In the human gastrointestinal tract, differential as well as absolute measurements of electrical potential within groups of cells can be made by piercing small electrodes through the inner gastrointestinal layer of tissue, the mucosa. Pressures can be measured by monitoring the amount of pressure required to keep fluid flowing slowly out the end of a tube placed where the measurement is to be made. Measurements of displacement can be made in animals by mounting transducers on the stomach or bowels. The electrical potential within individual cells can be observed by taking tissue samples and probing a particular type of cell with an electrolyte filled glass microelectrode. Measurements of forces generated by a piece of tissue can also be made through the use of strain gauges.

Before the signals from the various transducers can be used, they must be conditioned. This is done by selecting gains and cut off frequencies of amplifiers and filters in

available biomedical equipment. The signals then can be plotted, stored on analog magnetic tape, or otherwise monitored.

It is not sufficient to record only the conditioned signals from a number of transducers. Descriptive information must be recorded specifying the condition of the subject, when and how drugs have been administered, the time and date data were recorded, the transducers and calibration points used, where the transducers were placed, any special equipment used, and so on. Data can easily become meaningless if the precise conditions under which it was gathered are not described as thoroughly as possible.

To gain insight into the processes involved in gastrointestinal motility, the different types of activities monitored may be analysed. One way of reducing the volumes of data that can be acquired is to use the Fourier transform and to study the resulting frequency spectra. Correlations between the activities monitored are also sought, such as the relationship between the electrical activity at two locations or the relationship between pressure waves and electrical activity at the same location. Other possible relationships being investigated are how the activities within the gastrointestinal tract or its tissues vary under the influences of certain drugs or other conditions.

As this research is being carried out at the University of Alberta, it is felt that the above type of data analysis should be done on the University's powerful computer system,

based on the Amdahl 470V/8 computer operating under the Michigan Terminal System (MTS), rather than on a dedicated minicomputer system. Some advantages would be the availability of extensive software support including subroutine libraries, a number of high level languages, a powerful file editor, and access to resources such as disk space, magnetic tape drive units, and graphics plotting devices. Also, hardware, operating system, and some software maintenance are provided for. Using the University's computer is also convenient as there are a number of communication lines and terminals across campus providing access to the resources of this system.

But the question arises as to how the signals from the transducers and the descriptive information are to be transmitted to the large time shared computer system. It is the function of the data acquisition and transmission system described in this thesis to sample the conditioned analog signals and to transmit this digitized information, as well as any amount of descriptive information, to the computer system. This must be done in a quick, reliable, and convenient way.

II. Requirements

The requirements of the data acquisition and transmission system are to perform its function of signal sampling and data transmission with reliability, convenience, efficiency, and with as much flexibility as possible. As the system is to be used in a biomedical laboratory environment, it must be easy to operate, allowing the user to more fully concentrate on other aspects of his research. The system must be robust to allow for operator errors, for vibrations or mechanical shock, and for minimal servicing in the future. Reliable data acquisition and transmission are important as lost data may be difficult or impossible to regenerate. It is desirable that the system be as flexible as possible, to maximize its possible applications and give the user some freedom or control in the operation of the system.

In terms of function, the position of the data acquisition and transmission system in relation to its environment is indicated by the block diagram in Figure II.1. In order to perform its function, the system must contain a number of devices and processes as shown in Figure II.2

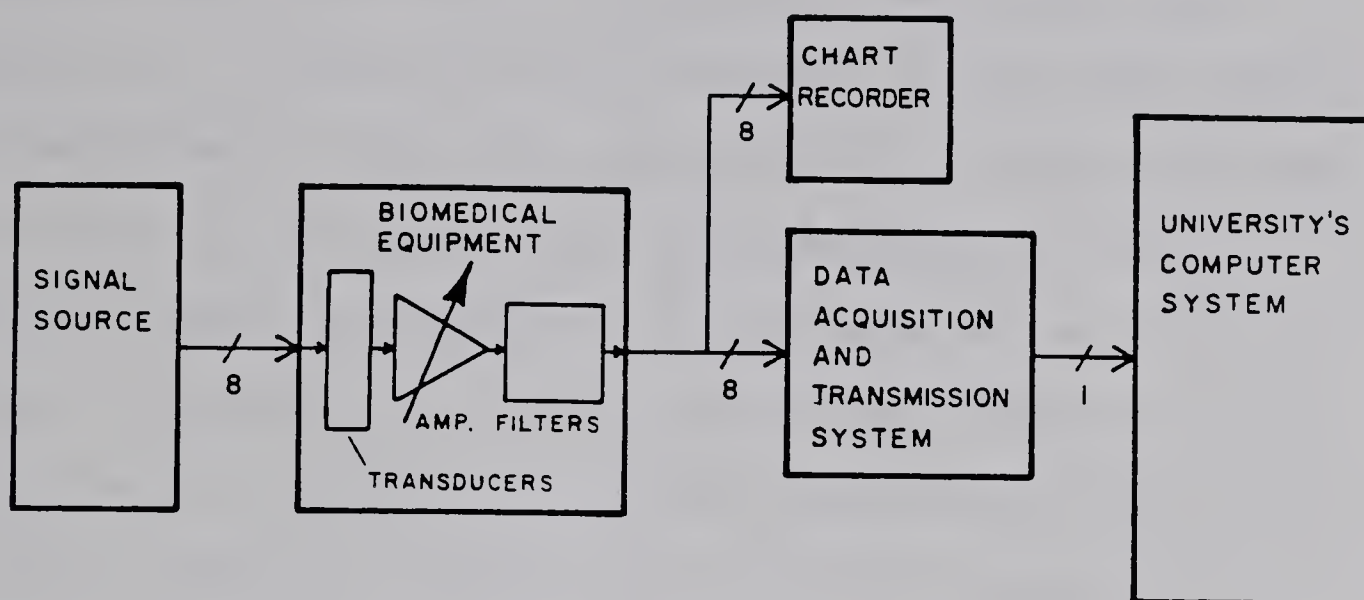


Figure II.1 The Environment of the Data Acquisition and Transmission System

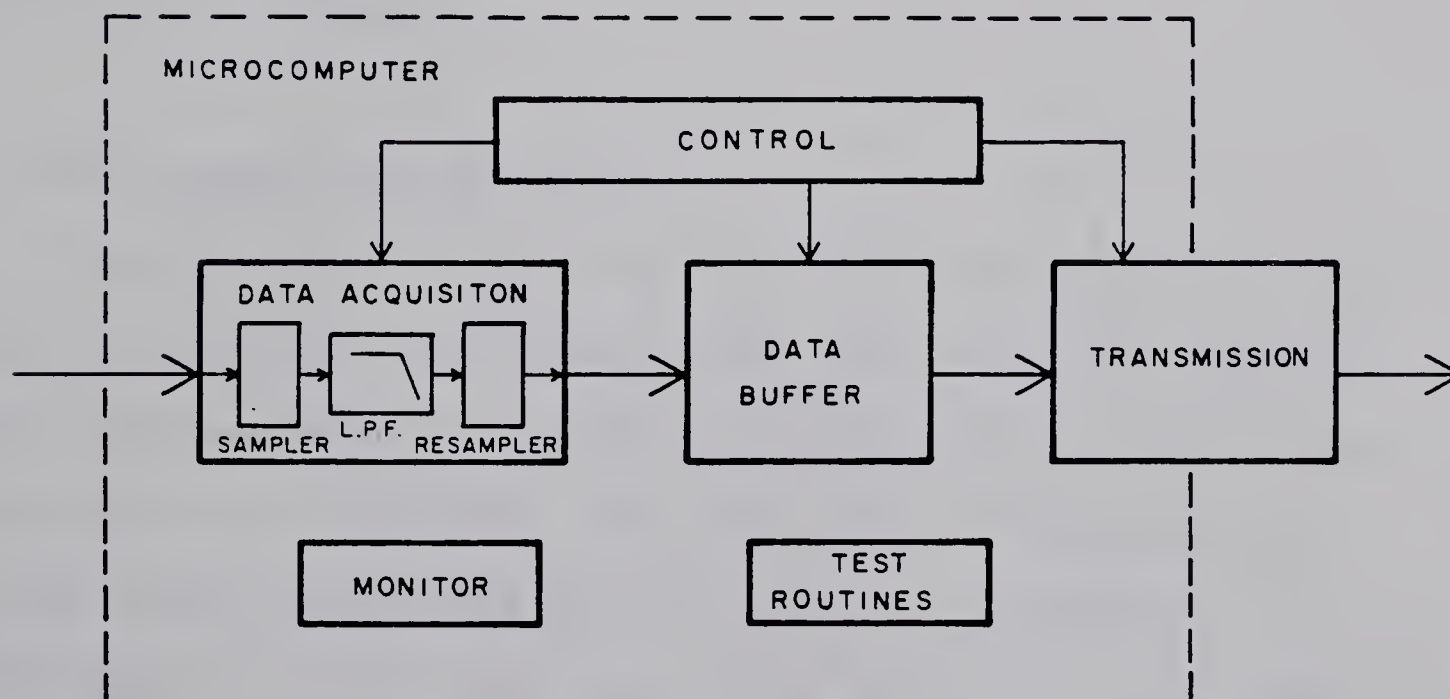


Figure II.2 The Data Acquisition and Transmission System

A. Microcomputer

A central device in the system is required to control and execute the functions of sampling, processing, and transmission. The use of a microcomputer allows this device to be dedicated to these functions with its design tailored to meet particular objectives. The microcomputer can be constructed and its components can be selected to achieve a resistance to abuse, a reduced need for maintenance, and a high degree of reliability. Software can be written with a particular function in mind to provide efficiency and can be structured to provide reliability and to allow for future changes. A data acquisition and transmission system based on a dedicated microcomputer would be easier to operate and less costly to obtain and maintain than a minicomputer based system or a general purpose commercial data acquisition and transmission system.

B. Data Acquisition Process

In the type of research described, two types of data must be acquired. One type of data may be called sampled data and consists of a series of sampled data points from the signals of interest. The other type of data may be called descriptive data and consists of a precise description of the conditions under which the first type of data was acquired. Descriptive data must be acquired and entered into the system by the operator. It is the function of the data acquisition process to obtain sampled data.

Biomedical data acquisition equipment is used to condition the signals from the transducers. This equipment is always adjusted to produce an output within the plus and minus two volt range. This is the range required for plotting this data on a chart recorder. ¹ This equipment can be adjusted to have a bandwidth limited to less than 20 hertz. As a consequence, each output signal of this equipment, which is the sum of the signal related to a particular activity under investigation plus any noise picked up, would have frequency components below 20 hertz. These conditioned signals are the ones sampled by the data acquisition process.

To avoid distortion, it is sufficient that the data acquisition process have a fixed input range of plus and minus two and a half volts and that the initial sampling frequency be at least 40 hertz. Depending on the particular activity being monitored, a signal derived from the gastrointestinal tract contains a maximum frequency component, that is of any significance, between .5 and 3 hertz. As a result, the data acquisition process should have a variable resampling rate of about two to ten hertz. When a lower resampling rate is sufficient, the volume of data acquired can be greatly reduced by allowing the user to select a lower resampling frequency.

¹In our case, the Beckmann Type R Dynograph Recorder is used to provide signal conditioning and chart recording for 8 channels.

Since this resampled data, (also referred to as sampled data), will later be analysed using the Fourier transform, it is important that its frequency spectra are not distorted. In order to avoid aliasing, prior to resampling data should be filtered to remove the appropriate higher frequency components entered either by the signal or by noise.

For our purpose, it would be sufficient if the data acquisition process could sample up to eight of the conditioned analog signals concurrently with a resolution of 12 bits.

C. Transmission Process

The transmission process is the mechanism by which the descriptive data and the sampled data are sent to the University's Amdahl computer system operating under the Michigan Terminal System. As most of the data will consist of sampled data, the transmission of sampled data should occur as fast as possible and should be concurrent with the data acquisition process. This would minimize the time the user must wait before this data is available for analysis. There must also be a means by which the user can enter and transmit descriptive data. The entry and transmission of descriptive data must not interfere with the transmission of sampled data.

As a large volume of data will be sent to a time shared

computer system, a handshaking mechanism ² is required to ensure that the transmission process sends data only when the computer system is ready to accept it. Without proper handshaking, data may be lost.

The means by which the time shared computer system can receive and store the two data types should be as economical and yet as flexible as possible. Also, any programs used to analyse the data must be able to differentiate between the two data types.

D. Data Buffer

When a significant amount of data is to be transmitted to a time shared computer system, it is difficult if not impossible to predict exactly when that system is ready to accept this data. The rate at which data may be transmitted to such a system is a function of how heavily the system is loaded, and this loading is usually dynamic and somewhat unpredictable. There is no guarantee that data may be transmitted when it is acquired or as fast as it is acquired. If the University's computer system should go down, it may even be a few hours before the system comes up again allowing transmission to resume. To overcome these problems, a data buffer is required within the data

² Handshaking has been defined as a mechanism whereby a sending device indicates the availability of data and the receiving device indicates the receipt of this data and/or a readiness to accept it.

Lance A. Leventhal, 6809 Assembly Language Programming
 Osborne/McGraw-Hill, Berkeley, 1981, p. 12-5

acquisition and transmission system to hold data until it can be transmitted.

The flexibility of the data acquisition and transmission system increases with the size of its data buffer. If the rate of data acquisition is greater than the rate of data transmission, then the larger the data buffer, the greater the time interval before the data buffer is filled to capacity and data acquisition must stop. Similarly, if the time interval over which data is to be sampled is fixed, then the larger the data buffer, the larger the allowable difference between the rates of data acquisition and data transmission.

Further flexibility can be provided if the buffer is a nonvolatile memory device. This device would retain any untransmitted data if A.C. power must be removed from the data acquisition and transmission system. It should be noted that all mentioned aspects of the data buffer that improve flexibility also improve reliability by reducing the chance of data loss.

Some other qualities of the data buffer that must be considered are read and write access times, data transfer times, ³ read and write error probabilities, mean time between failures and servicing, as well as sensitivity to characteristics of the physical environment such as .

³ See Section IV.D

mechanical shock or electromagnetic radiation. ⁴

E. Control

A control device in the data acquisition and transmission system is required to allow the user to manipulate the operation of the data acquisition process, the transmission process and the data buffer. As this device is the interface between the user and the system, it should prevent the user from making incompatible requests, provide some guidance when the user makes an error, and provide an acknowledgement when a request is granted.

Keys, used to control system operation, and a display, used to indicate system status, can be mounted on the front panel of the microcomputer. This would allow for quick and simple system control.

F. Other Devices

Other devices required in the data acquisition and transmission system are a monitor and a set of test routines. The monitor will be used to handle keyboard entries and drive the display. It should also provide a means to monitor, stop, and start program execution as an aid to system development. This system development portion of the monitor should be enabled only during development or testing to prevent interference with the acquisition or transmission of valuable data. Test routines are required to

⁴ See Section III.D

ensure that each device or process in the system is functioning properly. These routines may be executed using the monitor during system development or at any time the system's operation is to be checked.

III. Specifications

Once the general functions and requirements of the data acquisition and transmission system have been considered, one can select or specify the major components or operations necessary to realize those requirements.

A. Microcomputer

As a dedicated microcomputer system is required, a microcomputer based on the Motorola MC6800 microprocessor was chosen. This choice was based on the support available at the time system development began. The most significant factor in making this choice was the availability of a cross assembler. Other considerations were the availability of MC6800 based system components and a familiarity with the device. Had the appropriate cross-assembler been available, one of the more powerful microprocessors, such as the Motorola MC6809 or the Intel iAPX 86 would have been a better alternative, as in a single processor system, if a processor with higher performance is used, the system would also be capable of greater performance. Perhaps the high performance of the MC68000 would have justified the added cost and effort required for its implementation.

B. Data Acquisition Process

As outlined by the requirements, the operations that must be performed within the data acquisition process are initial sampling, filtering, and resampling. As it is also

required to sample up to eight signals concurrently, programming is simplified and execution time reduced if each of the operations of sampling, filtering, and resampling always handle data from all eight input signals at once.

Initial Sampling

To be compatible with the 20 hertz bandwidth of the signals being sampled, a 40 hertz initial sampling frequency was selected. One advantage of using this sampling frequency is that any 60 hertz noise picked up in the analog circuits from the power lines would, because of aliasing, appear as a 20 hertz signal after sampling. This is quite convenient as the low pass digital filter described next provides the greatest attenuation at 20 hertz.

Digital Filter

The initially sampled data must be digitally filtered in real time to attenuate all frequencies within the bandwidth of the input signals that are above one half the resampling frequency. To accomplish this a low pass filter must be used with a cut off frequency slightly below one half the resampling frequency to account for the filter's roll off. If the resampling frequency is changed, the cut off frequency must also be changed accordingly by selecting the appropriate set of filter coefficients. This should be done automatically when the resampling frequency is selected. A fourth order low pass Butterworth digital filter algorithm provides sufficient attenuation and a sufficiently sharp roll off. As the numerical computation required by a

digital filter uses a significant amount of microprocessing time, the algorithm is implemented using fixed point arithmetic in order to speed up processing.

Resampling

The output of the data acquisition process is the filtered and resampled data. The user is allowed to choose a two, four, eight, or ten hertz resampling rate. In addition to producing sets of eight data points, one for each signal sampled, the resampling process provides more information and a means of reliability checking by noting the time each set of samples was taken. One could then check if any samples were missing or out of sequence. The user can also associate flags or markers to certain samples to indicate special points in time, which is useful.

C. Transmission Process

As shown in Figure III.1, the transmission medium used to communicate to the University's Amdahl 470V/8 based system is a dedicated four wire telephone line, one of many such lines connecting terminal points on campus to the computing facilities. The microcomputer system is connected to this line through an asynchronous serial communications interface adapter, (the MC6850 ACIA, a microcomputer peripheral device), and RS232 buffers to produce a standard serial data interface.

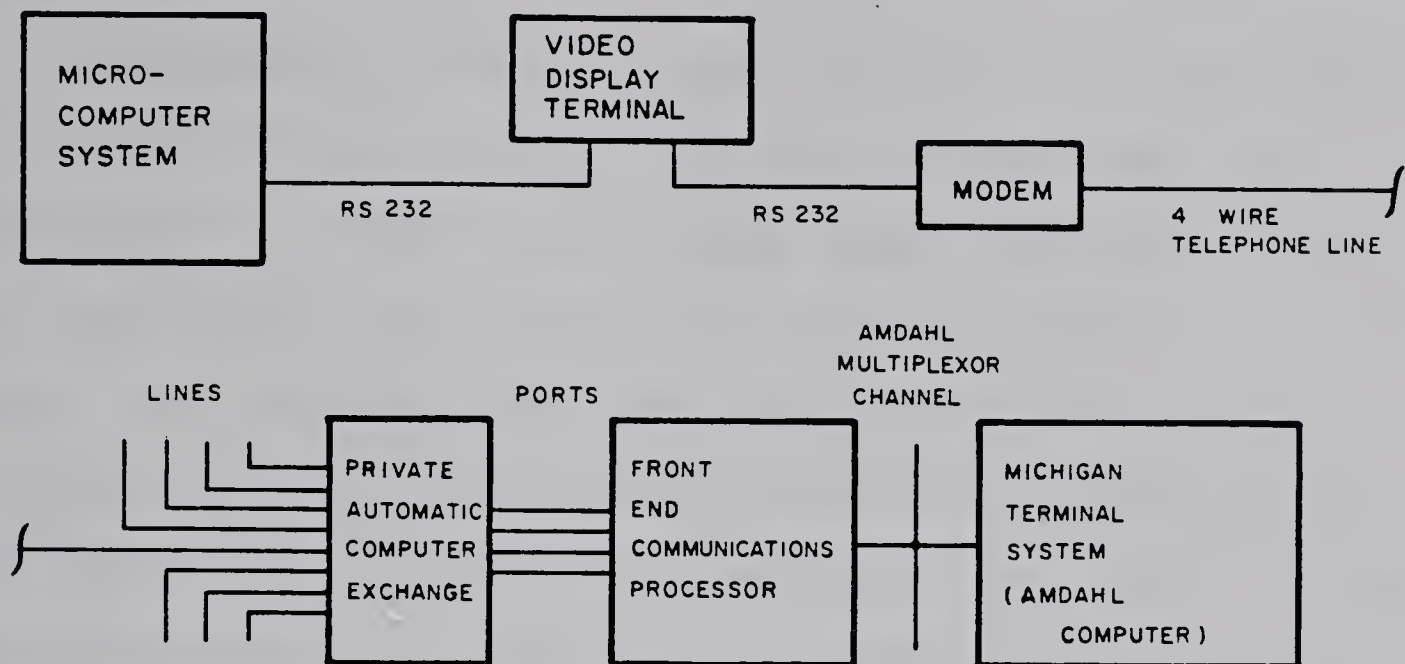


Figure III.1 Connection to the MTS System

This interface will connect the microcomputer to a video display terminal (VDT) and the terminal to a modem. The modem adapts the serial data transmission from RS232 to the signal modulation used on the four wire line. The serial data is coded in ASCII and is transmitted at 9600 baud with even parity and one stop bit. The front end communications processor and the terminal both operate in the full duplex mode. This means that an ASCII character sent by the microcomputer will be received and retransmitted by the terminal to be received by the Amdahl system's front end. Upon being received by the front end, the character will be sent back, (echoed), to be received by the terminal. The character is then displayed on the terminal and sent back to the microcomputer. Under the assumption that the



transmission medium is reliable and in order to save processor time, the microcomputer ignores the characters echoed back to it.

The handshaking method, used to ensure that the front end is able to receive data from the microcomputer, is implemented by activating the paper tape input mode in the front end and by the transmission interface software in the microcomputer system. When this mode is activated in the front end, an ASCII "DC1" control character is sent by the front end to the microcomputer indicating that the front end is ready to receive a string of characters. The transmission software will then send a line of sampled data to the Amdahl system. When the front end is ready to receive another string of characters, another "DC1" character is sent to the microcomputer.

The terminal displays all ASCII characters that originate from either the microcomputer or the Amdahl system, (except control characters), and serves as a monitor indicating how transmission is proceeding. When the microcomputer has no information to transmit, say before data acquisition has begun or after all acquired data has been transmitted, the terminal can be used by the operator to enter and transmit any type or amount of descriptive information. Minor restrictions must be imposed on the format of the descriptive data entered so that programs can distinguish it from sampled data. Also, lines of descriptive data should not be longer than lines of sampled data so that

data can be efficiently stored on magnetic tape. In this way, transmission of descriptive data is compatible with the paper tape input mode and does not interfere with the transmission of sampled data. The descriptive data entered and echoed back to the microcomputer is ignored.

All data transmitted to the Amdahl system is placed in a data file on disk. Two ways in which this can be done are through the use of the MTS \$COPY command or by running a program which reads data received from the microcomputer and places it into the file. The program used to receive this data may be written to execute, when requested, say by a flag, a certain type of analysis on any data already received. The results of this analysis could be transmitted back to the terminal, another device in series between the modem and the microcomputer, or a device connected to another line from the Amdahl computer system. Although a Fortran program was written to receive data and place it in a file, this type of analysis has not yet been implemented, but would be compatible to the transmission process as described.

D. Data Buffer

The device selected to implement the data buffer is the Intel 7110 one megabit bubble memory. This nonvolatile high density memory device requires no servicing, has a long life expectancy, and is tolerant of mechanical shock and

vibration.⁵ Intel Corporation appears determined to make the 7110 as reliable as possible. The 7110 is magnetically shielded to protect it from external magnetic fields. Samples undergo quality assurance tests at each stage in the production of the device. The devices produced undergo product testing and samples are taken for life tests and reliability stress tests. Stress testing includes exposure to steam, humidity, 150 degree Centigrade temperatures, shock, vibration, and temperature cycling.

The hardware required to incorporate the bubble memory within a microcomputer system is rapidly developed by obtaining the BPK 72 Bubble Memory Prototype Kit from Intel which provides the necessary interface devices and documentation. The 7110 interface devices include a controller chip, coil driver and pulse generator circuitry, and a serial data formatter and sense amplifier chip. The support devices control the operation of the bubble memory hardware including power up, power down, initialization, locating pages of data stored in the bubble memory, writing and reading data to and from the bubble memory, and automatic error detection and correction of data read from the bubble memory device. The controller chip controls the operation of most of the other devices and interfaces the bubble memory hardware to the rest of the microcomputer system. The microcomputer only has to supply instructions to

⁵John E. Davis, "The 7110, a One Megabit Magnetic Bubble Memory", **Reliability Report 22**, Intel Corporation, Santa Clara, September 1979

the controller chip, check its status, and be capable of transferring data to or from it up to a specific maximum rate. When automatic error detection and correction is enabled, read and write error rates have been calculated by Intel Magnetics Incorporated as being one bit in 10^{16} bits and one bit in 10^{20} bits respectively. ⁶ A single controller chip may handle one, two, four, or eight bubble memory devices simultaneously.

E. Control

A number of control functions are provided to allow the user to specify or manipulate the operation of the data acquisition and transmission system. The operator is able to stop and start the processes of data acquisition and data transmission. He is able to select a resampling frequency, a filter cut off frequency, and the use of the MTS \$COPY command or the use of a special program to receive, store, and process the transmitted data. The user is able to clear any data in the bubble memory data buffer or indicate that this data is to be retained during power down. Control functions allow the operator to calibrate sampled data, to attach a flag or a marker to a sample, and to set a real time clock in the microcomputer system so that the time when samples are taken can be noted. As a convenience, a function is available to automatically sign the user on to the MTS

⁶John E. Davis, "The 7110, a One Megabit Magnetic Bubble Memory", Reliability Report 22, Intel Corporation, Santa Clara, September 1979

operating system.

F. Other Devices

The system development functions performed by the monitor are to start program execution at a given location, to stop program execution, to display and allow for the alteration of the contents of any of the microprocessor's registers, to display and allow for the alteration of successive memory locations, to set a breakpoint, to move a block of data, and to execute simple bubble memory control functions. These functions are intended to aid software and hardware development and allow for the execution of test routines.

The test routines are used to check the operation of hardware and software modules within the microcomputer system. Test routines have been written to perform the functions of testing random access memory, checking the display hardware and software, testing the transmission of data, testing the writing and reading of large volumes of data to and from the bubble memory hardware and software system, and testing the data acquisition process.

IV. Implementation

In the design of a system, after one has specified the major components and processes that must work together to perform the required functions, one can devise ways in which these components and processes are to be implemented within the system.

A. Microcomputer System

As a dedicated microcomputer based on the Motorola MC6800 microprocessor was specified, a Motorola M68MM01A Monoboard Microcomputer was obtained to speed up system development. The Monoboard Microcomputer is a single printed circuit board which provides the one megahertz version of the MC6800 microprocessor, one kilobyte of static random access memory (RAM), sockets for read only memory (ROM), an RS232 asynchronous serial communications port (using the MC6850 ACIA), and an interface to a microcomputer system bus.

The other microcomputer system circuitry has been implemented on Motorola MEX68WW wire wrap boards and has been wire wrapped to provide high component densities, rapid construction, and a means to modify circuitry. Three such wire wrap boards have been developed and each is interfaced to the microcomputer system bus. One board provides data acquisition circuitry and a real time clock (the MC6840 PTM). Another board provides a multiplier circuit (interfaced through MC6821 PIA's) and the bubble memory

hardware. A third board provides extra random access memory, extra read only memory, an address trap for the monitor, a programmable timer (PTM), address decode circuitry, and an interface (PKDI) to a keypad and display board. The keypad and display circuitry has been wire wrapped on a single epoxy-glass Vector (registered trade mark) board mounted behind the front panel of the microcomputer system.

A block diagram of the microcomputer system's electronics is shown in Figure IV.1. This is the hardware necessary to execute the various processes within the data acquisition and transmission system.

The microcomputer system chassis forms a card cage into which the Monoboard Microcomputer and the wire wrap boards can be plugged to make contact with the microcomputer system bus. Provisions have been made for mounting the front panel, a back panel, a commercial power supply, and a printed circuit voltage regulator board that has been developed. The complete assembly slides inside a small Hammond (trade mark) cabinet.

As the microcomputer must perform a number of processes concurrently, a means of controlling the execution of software modules is required. An interrupt control scheme is sufficient, and is also efficient in terms of processor time, although the use of a multitasking operating system would likely be more flexible. With this interrupt control scheme, hardware is used to drive software. When the hardware associated with a certain process requires

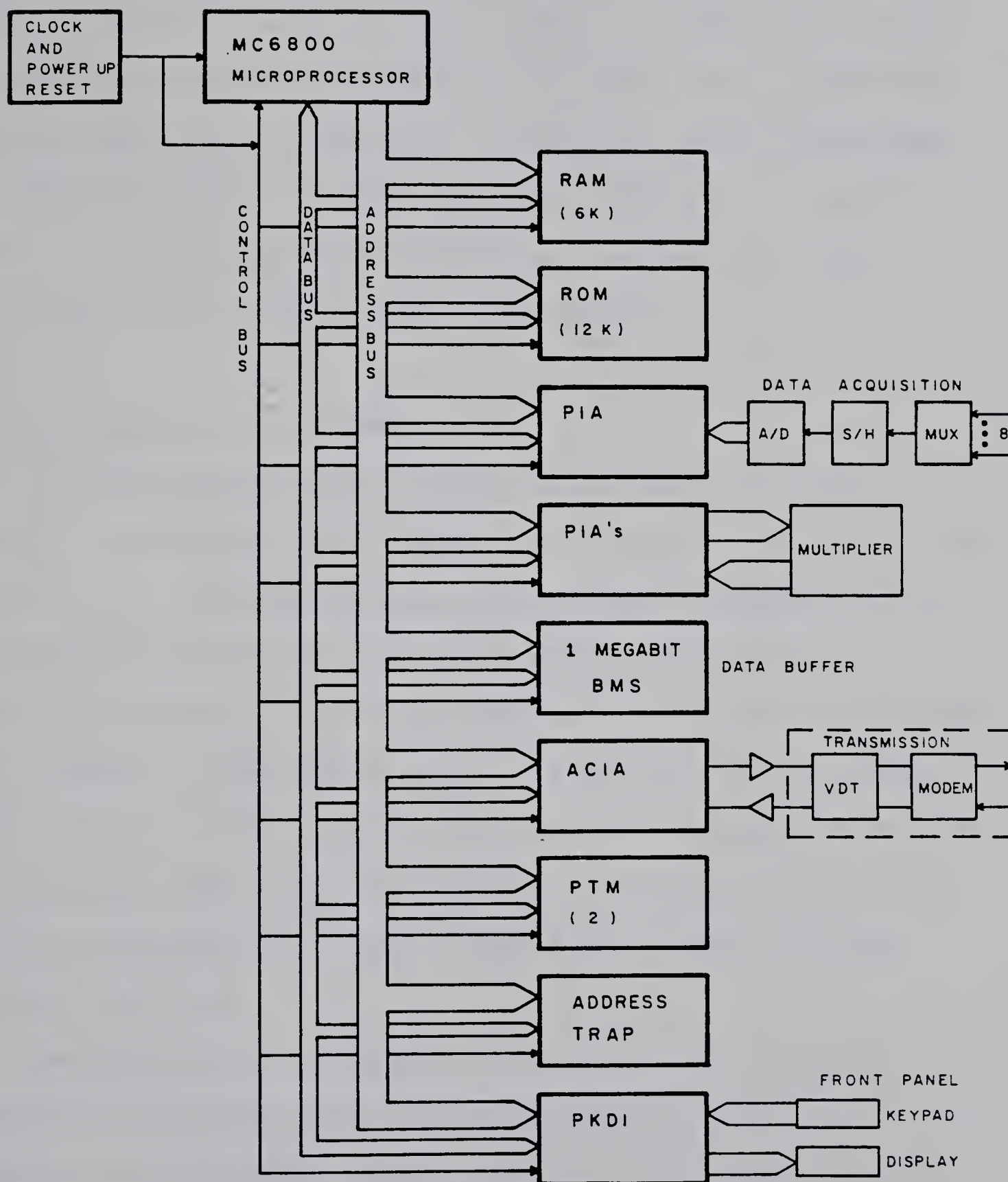


Figure IV.1 The Microcomputer System

servicing, it generates an interrupt request. A segment of code is executed to determine the source of the interrupt. This segment of code polls the interrupt sources in the order of their priority, higher priority interrupts first. As soon as an interrupt source is located, the appropriate service routines are executed. If another source generates an interrupt request at the the same time or at a later time, it will be serviced as soon as the service routines executing first allow further interrupts.

B. Data Acquisition Process

The data acquisition process is driven by an interrupt from a programmable timer module that occurs at a rate of 40 hertz. This interrupt source is the second highest priority in the microcomputer system and provides the initial sampling frequency. The data acquisition process is executed by a segment of code which calls a sampling routine named SAMP, calls a filter routine called FILT, and then resamples the acquired data. Routines are then called to form a line of data and store this line in the bubble memory system.

Initial Sampling

The eight analog signals to be sampled by the data acquisition process enter the microcomputer system via a plug on the back panel and are buffered with operational amplifiers set to gain of -4. As shown in Figure IV.2, these buffered signals enter an analog multiplexer, (MUX), that connects one of them at a time to a sample and hold device.

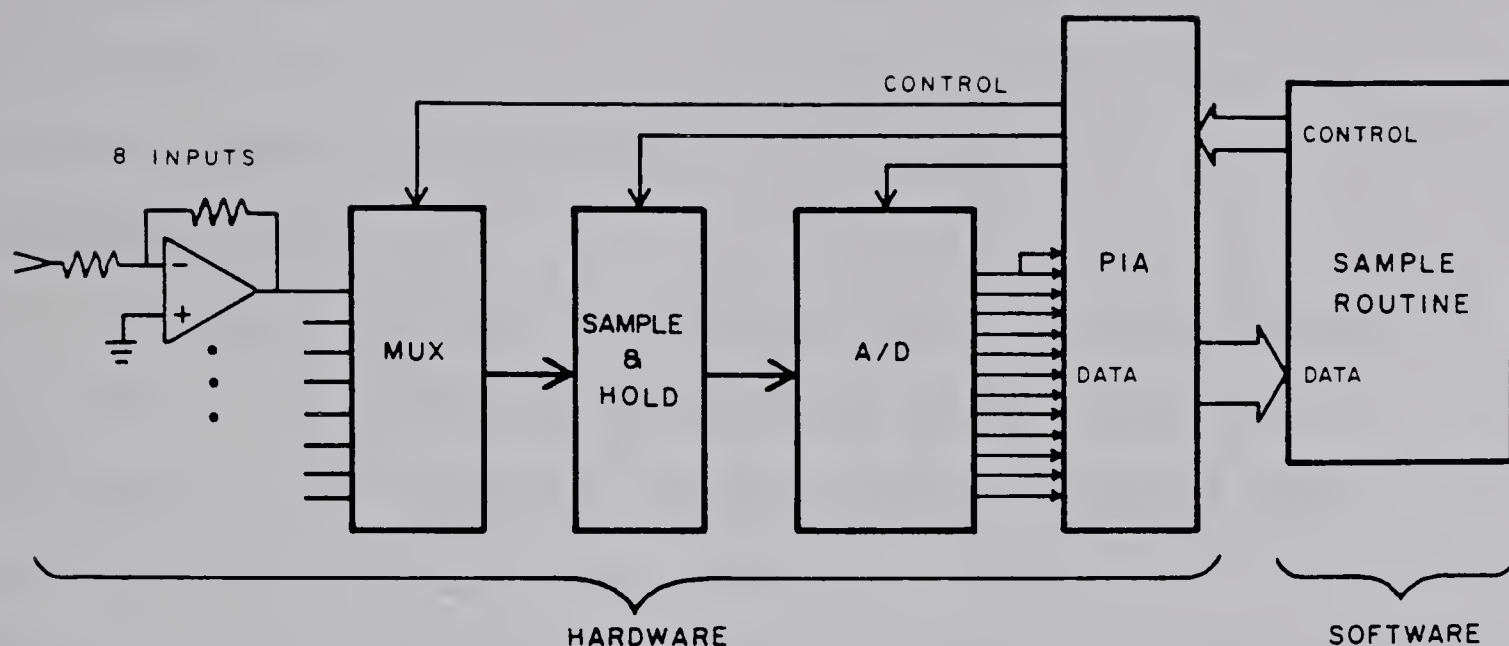


Figure IV.2 Implementation of Initial Sampling

This device will sample the signal and then maintain the sampled voltage level while an analog to digital converter (A/D) translates the voltage level into a 12 bit digital value. An MC6821 PIA parallel input and output device outputs control signals for the multiplexor, the sample and hold device, and the analog to digital converter and allows the 12 bit digital value to be read.

To obtain its input, the sampling routine controls the data acquisition hardware and reads the 12 bit digitized sample through the parallel input output device. The output of the sampling routine is a block of eight words, one word for each signal sampled. Each word consists of a 12 bit digital sample which is padded by extending the sample's most significant bit by two extra bits and by adding ten

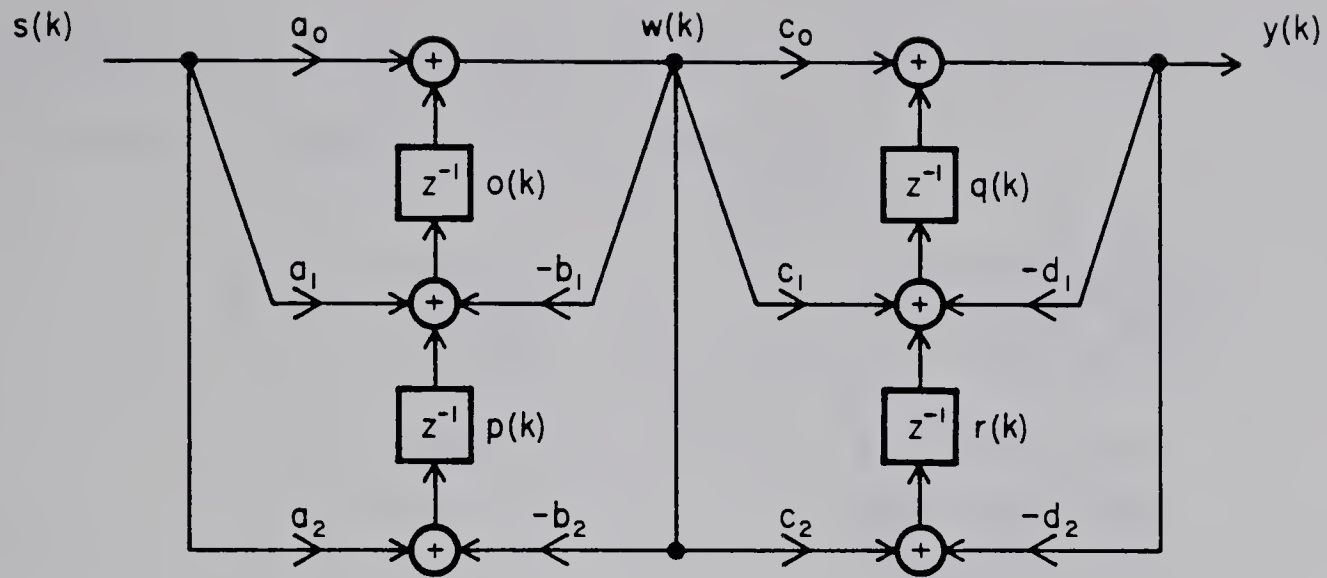
zero least significant bits. This is done to make the initially sampled data compatible with the fixed point arithmetic used in the digital filter algorithm and allows the values obtained to be increased by a factor of four without causing an overflow.

Digital Filter

An eight channel fourth order low pass Butterworth digital filter has been implemented using fixed point arithmetic. The assembly language source code has been written for one channel and then duplicated with the appropriate changes for the other channels to allow the code to be written in a form that executes as fast as possible. As Figure IV.3 shows, the fourth order transfer function has been realized using a cascade of two second order transfer functions.⁷

A Fortran program has been written (see Appendix C) to compute the filter coefficients A0 to D2 for a given sampling period and cut off frequency. Expressions for these coefficients have been derived from the normalized fourth order analog low pass Butterworth transfer function by using the bilinear transformation,

⁷ A cascade of second order functions reduces coefficient sensitivity problems.
 H.T. Nagle Jr., V.P. Nelson, "Digital Filter Implementation on 16-Bit Microcomputers", *Micro*, IEEE Computer Society, Vol. 1, No. 1, Feb. 1981, pp. 23-41



a. Filter Structure

$$\frac{Y(z)}{S(z)} = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}} \frac{c_0 + c_1 z^{-1} + c_2 z^{-2}}{1 + d_1 z^{-1} + d_2 z^{-2}}$$

b. Transfer Function

W = O	O = O + W*(B1+1)	where
O = P	P = P + W*B2	A0 = a0
Y = Q	Y = Y + W*C0	A1 = a1 B1 = -1 - b1
Q = R	Q = Q + W*C1	A2 = a2 B2 = -b2
R = P = 0 (zero)	R = R + W*C2	C0 = c0
W = W + S*A0		C1 = c1 D1 = -1 - d1
O = O + S*A1	Q = Q + Y*(D1+1)	C2 = c2 D2 = -d2
P = P + S*A2	R = R + Y*D2	

c. Fortran Code

Figure IV.3 Digital Filter Implementation

$$s \rightarrow \frac{2}{T} \frac{z-1}{z+1}$$

to arrive at a low pass digital filter prototype.⁸ Then the low pass to low pass transformation,

$$z^{-1} \rightarrow \frac{z^{-1} - a}{1 - az^{-1}}, \quad a = \frac{\sin(\frac{\theta_p - \theta_c}{2})}{\sin(\frac{\theta_p + \theta_c}{2})},$$

where

$$\begin{aligned} \theta_p &= 2 \tan^{-1}\left(\frac{T}{2} \omega_c\right) \\ \omega_c &= 1 \text{ rad/sec} \\ \theta_c &= 2\pi T(\text{desired cut off frequency}) \\ T &= \text{sampling period}, \end{aligned}$$

was used to arrive at the final digital filter transfer function having unity gain for a D.C. input.

In the fixed point arithmetic used, constants must have a magnitude of less than unity. But the low pass digital filter transfer function has the constants b1 and d1 with magnitudes between -1 and -2. To overcome this, B1 and D1 are used as defined in the figure.

To speed up numerical computation, a 16 bit by 16 bit hardware multiplier chip, the TRW MPY-16HJ, is used to perform multiplication. The routines, MF21 and MF22, which control this multiplication, multiply the data in two arbitrary memory locations and add the result to a third location, hence the structure of the Fortran code. The

⁸ This procedure is recommended in the literature. Harry Y-F. Lam, Analog and Digital Filters: Design and Realization, Prentice-Hall, Englewood Cliffs, 1979, pp. 543-558

multiplier chip provides a 32 bit product and it was found that the characteristics of the digital filter improve greatly if the additions of all products are carried out using 24 bit precision.

The sets of filter coefficients used are given in Table IV.1. The peculiar cut off frequencies were chosen to produce a set of coefficients that can be rounded to 16 bit fixed point format without significantly changing the filter transfer function's pole and zero locations in the Z-plane.

Table IV.1 The Sets of Filter Coefficients

	FILTER	CUT OFF FREQUENCY	RESAMPLING RATE	FILTER COEFFICIENTS			
				DECIMAL		HEXADECIMAL	
				A, B	C, D	A, B	C, D
A0, C0	02	(HERTZ) .600	(HERTZ) 2	.002136	.002045	0046	0043
A1, C1				.004272	.004089	008C	0086
A2, C2				.002136	.002045	0046	0043
B1, D1				.921906	.831848	7601	6A7A
B2, D2				-.930481	-.840027	88E6	947A
A0, C0	04	1.225	4	.008606	.007843	011A	0101
A1, C1				.017212	.015686	0234	0202
A2, C2				.008606	.007843	011A	0101
B1, D1				.829224	.668335	6A24	558C
B2, D2				-.863617	-.699707	9175	A670
A0, C0	08	2.450	8	.031982	.027161	0418	037A
A1, C1				.063965	.054321	0830	06F4
A2, C2				.031982	.027161	0418	037A
B1, D1				.620850	.376343	4F78	302C
B2, D2				-.748749	-.484955	A029	C1ED
A0, C0	0A	3.0627	10	.048218	.039764	062C	0517
A1, C1				.096435	.079529	0C58	0A2E
A2, C2				.048218	.039764	062C	0517
B1, D1				.506256	.242004	40CD	1EFA
B2, D2				-.699127	-.401031	A683	CCAB

Resampling

Resampling is accomplished simply by counting a specified number of sampling process executions and taking

the output of the eight channel digital filter every time this number is reached. After resampling, a 24 byte line of data is formed which consists of the eight output words of the filter rounded to 16 bits, a 16 bit check sum which is the sum of these 8 words, the time the resample was taken in hours, minutes, and seconds, a one byte flag or marker, and front and end line framing bytes. The current time is obtained from a programmable timer module, while the flag or marker is entered by the user via a control function. The line of data is then stored within the bubble memory system.

C. Transmission Process

A general description of the transmission process, including the hardware and the entry of descriptive data required by the operator, has already been presented.⁹ This section will describe the implementation of parts of the transmission process in more detail.

Transmission Interface Routines

The transmission process within the microcomputer system is controlled so that, logically speaking, a line of characters is transmitted at a time. There are two types of lines that can be sent to the MTS system, instruction or control lines which request some action, and data lines which must be stored by the MTS system. Consequently there are two sets of routines to effect this transmission. The routines STX1, TX1, and RX1 are used to transmit

⁹See Section III.C

instructions while the routines STX2, TX2, and RX2 are used to transmit data. As the two sets of transmission interface routines have the same structure, only the structure ^{10 11} of the first set is shown in Figure IV.4. Both sets of routines receive and transmit characters through the same asynchronous communications interface adapter, the MC6850 ACIA, and only one set may be active at any one time.

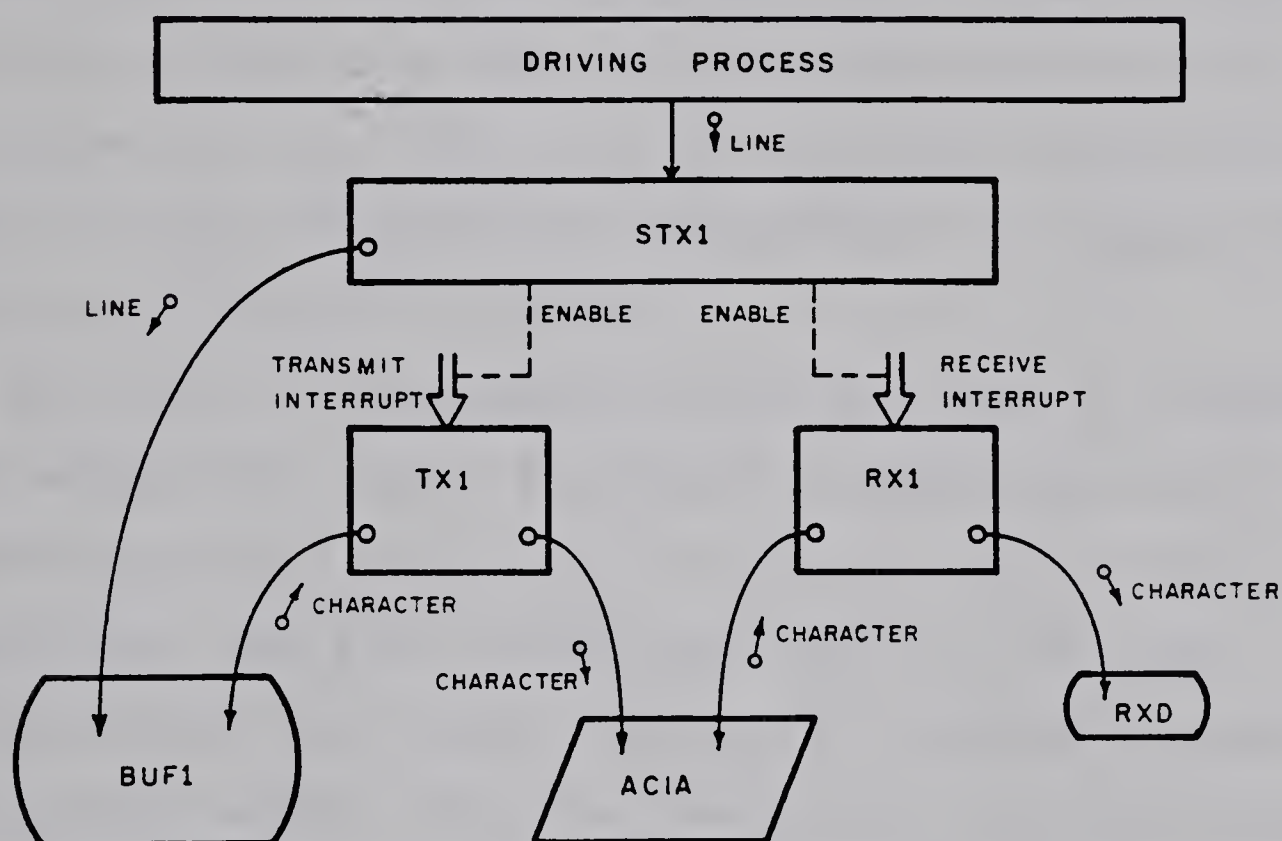


Figure IV.4 Structure of the Transmission Interface Routines

¹⁰ See the following for an explanation of program structure.

Judith C. Enos and R.L. Van Tilburg, "Tutorial Series - 5: Software Design", *Computer*, IEEE Computer Society, Vol. 14, No. 2, Feb. 1981, pp. 61-83

¹¹ Glenford J. Myers, *Composite/Structured Design*, Van Nostrand Reinhold, New York, 1978

To send an instruction or a data line, the driving process will call STX1 or STX2 and provide a string of characters to be transmitted. STX1 or STX2 will then place this line in a small buffer and enable the ACIA interrupts. Interrupts from the ACIA occur whenever it can transmit a character or when it has just received a character. Upon the interrupt request, a TX or RX routine is called to transmit or receive a character. The ACIA interrupt is given the highest priority within the microcomputer system to reduce its latency time ¹² so that no characters to be received by the system are lost. While one line is being transmitted, a number of other lines may be placed within the appropriate buffer by calling STX1 or STX2.

STX1 and its associated routines are used to transmit lines without the use of any specific acknowledgement or handshaking scheme being used to indicate that MTS has received the line and is ready for the next line. The calling process must ensure that the MTS system is capable of accepting another line. The routine RX1 provides a means of doing this by receiving all characters sent to the microcomputer by the MTS system and placing any special characters received in the location RXD. When the calling process sees that the appropriate character is in the location RXD, the next line may be sent. The routine TX1

¹² Latency time has been defined as the time from when a device generates an interrupt request and the time when that device is serviced.

G. Jack Lipovski, *Microcomputer Interfacing* Lexington Books, D.C. Heath & Company, 1980, p. 11

transmits all the data in BUF1 until this buffer is empty. This set of routines is called by processes such as the function to signon to the MTS system and the routine to transmit a set of commands to start or stop the transmission of sampled data.

STX2 and its associated routines are used to transmit sampled data with the paper tape input handshaking feature using the ASCII control character, "DC1". The calling process must only supply data lines while these routines handle the handshaking procedure with the front end communications processor of the MTS system so that no lines of data are transmitted at the wrong time. When STX2 places data in BUF2, the buffer associated with these routines, it enables the ACIA transmission interrupt. Driven by the ACIA transmission interrupt requests, TX2 takes ASCII characters from BUF2 and transmits them until a complete line is sent. After a complete data line is transmitted, TX2 disables the ACIA transmission interrupt and enables the ACIA receive data interrupt. When the front end communications processor sends a "DC1" character to indicate it is ready to receive another line, the RX2 routine will enable the ACIA transmission interrupt and disable the ACIA receive data interrupt. This causes the transmission of the next line of data to begin. While the transmission of a data line is taking place, STX2 may be called a number of times to place a number of data lines in BUF2.

As the terminal and the front end communications processor of the MTS system are in the full duplex mode, any character sent by the microcomputer system is echoed back to it. This provides a means for the transmission interface routines to check if that character has passed through the transmission hardware unaltered. But in order to save processor time and speed up the transmission process, this checking is not done. The assumption is made that the transmission of ASCII characters to the MTS system is totally reliable.

Transmission Process Driver

The process used to drive the transmission of sampled data is a mainline routine. Being the lowest priority process within the microcomputer system, it is interrupted by all other processes. This process is started when the user requests the transmission of sampled data and is terminated when this transmission is completed or when the user requests this process to stop. During this process, STX2 is called to transmit each line of sampled data. A flowchart of this process is shown in Figure IV.5.

Data Format

The data acquisition and transmission system organizes and controls data flow as lines of data. The data acquisition process produces lines of data containing a sample of each of the eight filtered input signals, a check sum, the time these samples were taken, and a flag or marker byte. These lines are stored in the data buffer until

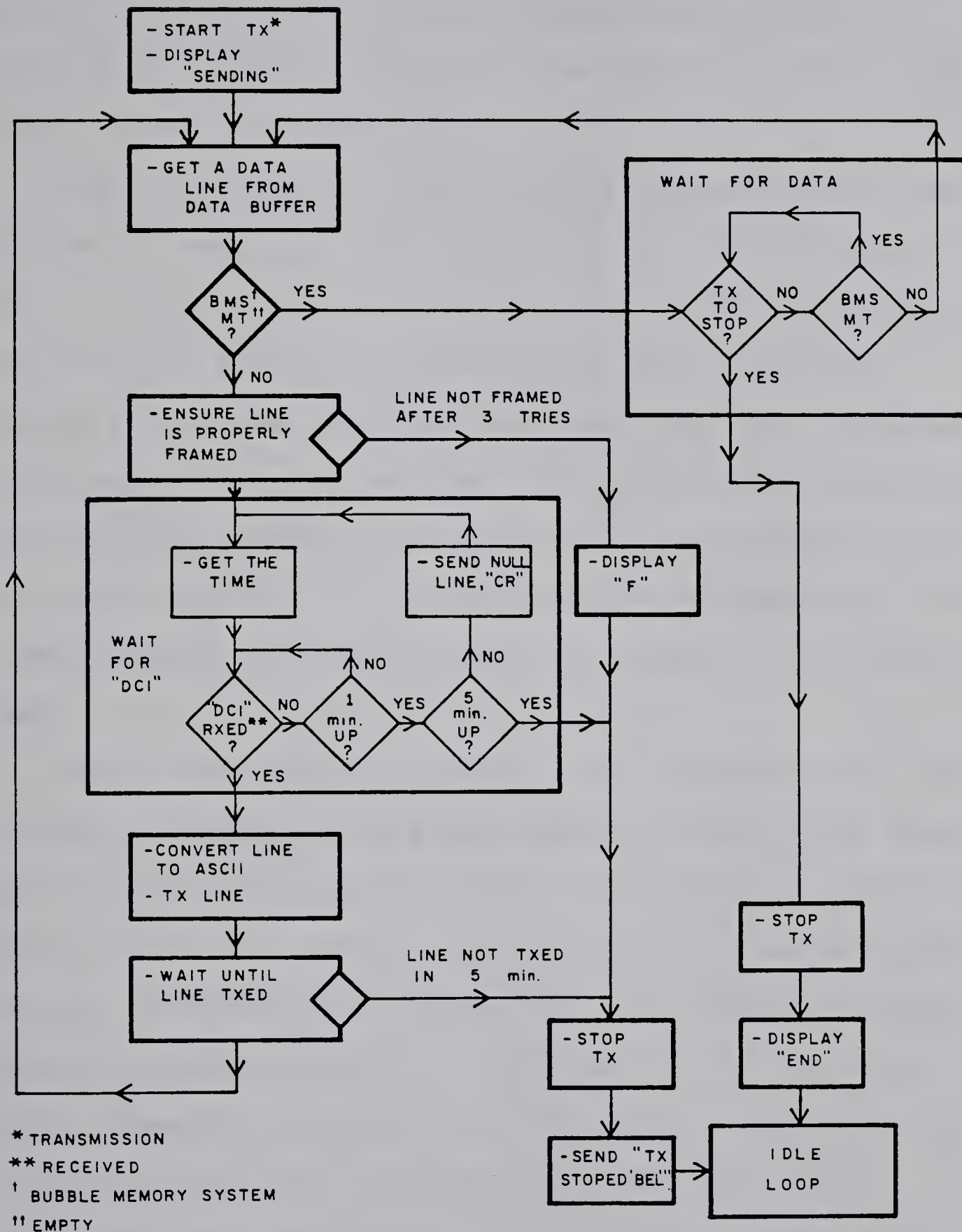


Figure IV.5 The Transmission Process Driver

the transmission process removes them from the buffer, converts them to ASCII, and transmits them. These lines of data are then received by the MTS system and stored in a file on disk. Figure IV.6 shows an example of a small amount of data stored in a disk file.

Lines of descriptive data that have been entered from the terminal when the transmission driver was in the waiting state all begin with a "1" in the first column provided the line is within a block of descriptive data. The other characters in these lines are arbitrary, but the line should not be longer than 45 characters. Blocks of descriptive data may contain any number of lines but must be preceded by a line containing only a "1" and should be followed by a line of descriptive data having a zero or a blank in the first column.

Lines of sampled data always start with a blank. The following 32 characters are the samples of the eight input signals coded in hexadecimal form. The first four characters represent the first sample, the next four the second sample, and so on. Following the samples are four characters which represent a check sum which is the sum of the previous 8 samples formed by unsigned 16 bit fixed point addition. The following 6 characters represent the time the line of sampled data was taken. Hours, minutes, and seconds are represented by two decimal characters each. The last two characters are the flag or marker byte and is coded in hexadecimal.


```

1      1
2      1  A TEST SIGNAL
3      1  CHANNEL 1 CALIBRATED AT 1.000 V AND
4      1  -.0006 VDLTS
5      1  A .2 HZ. SQUARE WAVE BETWEEN THE ABOVE
6      1  LEVELS
7      1  2 HERTZ RESAMPLING FREQUENCY USED
8
9      000200020003000200030004000300030016101655C1
10     OCC30CAE0CD30CA50CAE0CB60CBA0CA665B3101718D1
11     0000000000000000000000000000000000000000000102031F0
12     000000000000000000000000000000000000000000010203200
13     00010001000200010001000100020001000A10203200
14     00020001000300010001000200030001000E10203300
15     00030002000300020002000300030002001410203300
16     015501520155015001520154015401500A6610203400
17     065609460964094406460651065209444A7A10203400
18     0E170DFD0E270DF80DFD0E0B0E0D0DF5703D10203500
19     0D150CFD0D240CF60CFD0D0A0D0B0CF6553710203500
20     0C5F0C460C6D0C440C490C540C550C43625E10203600
21     0C0C0BF60C160BEF0BF50C000C020BF05FF110203600
22     04D304CA04D504C704CB04CE04CF04C5255C10203700
23     FEDCFEDFFEDBFEDFFEDFEDEDDFEDDFEDFF6EC10203700
24     FF66FF68FF68FF6AFF6AFF68FF6AFF68FF68110203800
25     00640063006500630063006400630064031D10203500
26     00620061006200600061006100610061030910203900
27     068F06820695057F0652068905560650343610203600
28     0D820D6C0D920D650D6B0D770D750D636BA210204000
29     0D970D7F0DA60D750D7F0D8C0D5C0D786C4310204000
30     0C650C600C730C4A0C500C5C0C5D0C4A62C510204100
31     0C840C6F0C630C650C6F0C7A0C7C0C6963BC10204100
32     075F0753079607500755076C075D07813C4A10204200
33     FFD1FFCFFFD1FFD0FFD0FFD2FFD2FFD2FE5710204200
34     FEEEFEEFFEEFDEEFEEFFEEFFEEFFEEFFEF1F77710204300
35     00490045004600470045004800450045024110204300
36     00390035003600390039003A003A003901C910204400
37     03E603DE03E603DD03DF03E203E303DE1F0C10204400
38     0C2B0C170C350C100C170C210C220C0F60F310204500
39     0E030DEB0E130DE40DE90DF90DF60DE36FA510204500
40     0C960C7F0CA40C750C800C8C0C8D0C79644310204600
41     0C7A0C650C880C6E0C660C700C720C5F636C10204600
42     0A0009F00A0D09E06F009FA06F809E94FB410204700
43     018B0156018C01550187015A018A01870C4410204700

```

Figure IV.6 Example of the Data Format

This byte is used to indicate something special about the sample or the time the sample was taken. During transmission, each line is followed by an ASCII carriage return character. Each line of sampled data appears on the terminal in the above format when it is transmitted.

Routines have been developed to read the data formatted in this way from a data file and to calibrate any sampled data read. A program using these routines (see Appendix C) has produced the output shown in Figure IV.7 from the data in Figure IV.6.


```

PRINTOUT FROM DATA FILE TEST2

FROM LINE      1
TO LINE 1000000

#1      #2      #3      #4      ***      #7      #8      FLAG      TIME      LINE NUMBER
VOLTS    VOLTS    VOLTS    VOLTS                                VOLTS    VOLTS

A TEST SIGNAL
CHANNEL 1 CALIBRATED AT 1.000 V AND
- 0009 VOLTS
A 2 HZ SQUARE WAVE BETWEEN THE ABOVE
LEVELS
2 HERTZ RESAMPLING FREQUENCY USED

-0.0009  -0.0003  0.0000  -0.0003  0.0000  0.0000  C1  10:16:55  10
 1.0000  0.9911  1.0024  0.9893  0.9948  0.9887  D1  10:17:18  11
-0.0015  -0.0009  -0.0009  -0.0009  -0.0009  -0.0009  FO  10:20:31  12
-0.0015  -0.0009  -0.0009  -0.0009  -0.0009  -0.0009  00  10:20:32  13
-0.0012  -0.0006  -0.0003  -0.0006  -0.0003  -0.0006  00  10:20:32  14
-0.0009  -0.0006  0.0000  -0.0006  0.0000  -0.0006  00  10:20:33  15
-0.0006  -0.0003  0.0000  -0.0003  0.0000  -0.0003  00  10:20:33  16
 0.1030  0.1024  0.1033  0.1018  0.1030  0.1018  00  10:20:34  17
 0.7321  0.7256  0.7338  0.7240  0.7283  0.7240  00  10:20:34  18
 1.1042  1.0935  1.1064  1.0920  1.0984  1.0911  00  10:20:35  19
 1.0261  1.0163  1.0272  1.0141  1.0196  1.0131  00  10:20:35  20
 0.9593  0.9503  0.9713  0.9587  0.9639  0.9584  00  10:20:35  21
 0.9439  0.9349  0.9456  0.9328  0.9386  0.9331  00  10:20:36  22
 0.3771  0.3738  0.3781  0.3729  0.3753  0.3732  00  10:20:37  23
-0.0910  -0.0892  -0.0904  -0.0892  -0.0898  -0.0892  00  10:20:37  24
-0.0478  -0.0464  -0.0470  -0.0467  -0.0467  -0.0464  00  10:20:38  25
 0.0291  0.0294  0.0300  0.0294  0.0294  0.0297  00  10:20:38  26
 0.0285  0.0267  0.0291  0.0284  0.0287  0.0287  00  10:20:39  27
 0.5132  0.5083  0.5141  0.5073  0.5104  0.5076  00  10:20:39  28
 1.0585  1.0492  1.0608  1.0471  1.0529  1.0466  00  10:20:40  29
 1.0650  1.0560  1.0669  1.0629  1.0590  1.0529  00  10:20:40  30
 0.9712  0.9624  0.9731  0.9606  0.9664  0.9606  00  10:20:41  31
 0.9807  0.9719  0.9829  0.9697  0.9769  0.9700  00  10:20:41  32
 0.6917  0.6868  0.6936  0.6869  0.6899  0.6862  00  10:20:42  33
-0.0169  -0.0159  -0.0153  -0.0166  -0.0160  -0.0160  00  10:20:42  34
-0.0866  -0.0843  -0.0849  -0.0843  -0.0843  -0.0837  00  10:20:43  35
 0.0208  0.0211  0.0214  0.0208  0.0211  0.0211  00  10:20:43  36

```

Figure IV.7 Print Out of Example Data File

D. Bubble Memory System

To incorporate the bubble memory kit selected into the microcomputer system required some extra hardware. A 4 megahertz 50% duty cycle square wave signal source was required as well as some logic to make the interrupt features of the Intel bubble memory controller chip compatible with the interrupt mechanisms of the Motorola based microprocessor system.

To create a bubble memory system, a software package was developed to drive the bubble memory hardware. The nature of this bubble memory system is that of a large nonvolatile first in first out data buffer. It was intended that the software package be as easy to use as possible and

that its internal operation be transparent to the user or the calling routines.

Much of the internal control of the software package is based on interrupt mechanisms rather than on polling mechanisms. Interrupts are used to control the bubble memory system to minimize the share of the microprocessor's time that must be spent in executing bubble memory system routines. Interrupts from the bubble memory controller chip are given the third highest priority in the microprocessor system and an attempt has been made to minimize the execution time of program segments which do not allow further interrupts. The interrupt mechanisms used within the bubble memory system must allow for an ACIA interrupt latency time not greater than one millisecond. This is required to prevent the loss of control characters being received by the transmission process.

A number of specifications and constraints have been imposed on the software used to implement the bubble memory system. All data transferred to or from the bubble memory hardware must pass through a small 40 byte first in first out data buffer within the controller chip. When this buffer is half full, a data request interrupt is generated by the controller chip. When data is being transferred, the microcomputer system must be able to provide for a maximum averaged transfer rate of 12.5 kilobytes per second. If the 40 byte buffer should overflow or become empty during a transfer, this operation will fail. The bubble memory system

has been set up for a page size of 64 bytes. Data transfers to or from the bubble memory device must be in complete pages. The maximum access time to any page within the bubble memory device is 81 milliseconds.

The structure of the bubble memory system software is shown in Figure IV.8.

When a line of data is to be stored in the bubble memory system, the bubble memory system input routine, BMSI, is called. This routine takes the line of data and places it in the bubble memory input buffer, BMIB, located in random access memory. When a complete page of data has been entered, BMSI enables the bubble control interrupt from the controller chip and indicates that a page write request has been made.

When the bubble controller chip becomes idle, a bubble control interrupt occurs and the bubble control interrupt handler, BCIH, sees the write request and calls BMWS which starts the transfer of a page to the bubble controller chip. The start of this transfer involves sending a write instruction with parameters indicating which page in the bubble memory device is to be overwritten and sending the first 14 bytes of the page to the controller chip. The bubble data request interrupt is then enabled.

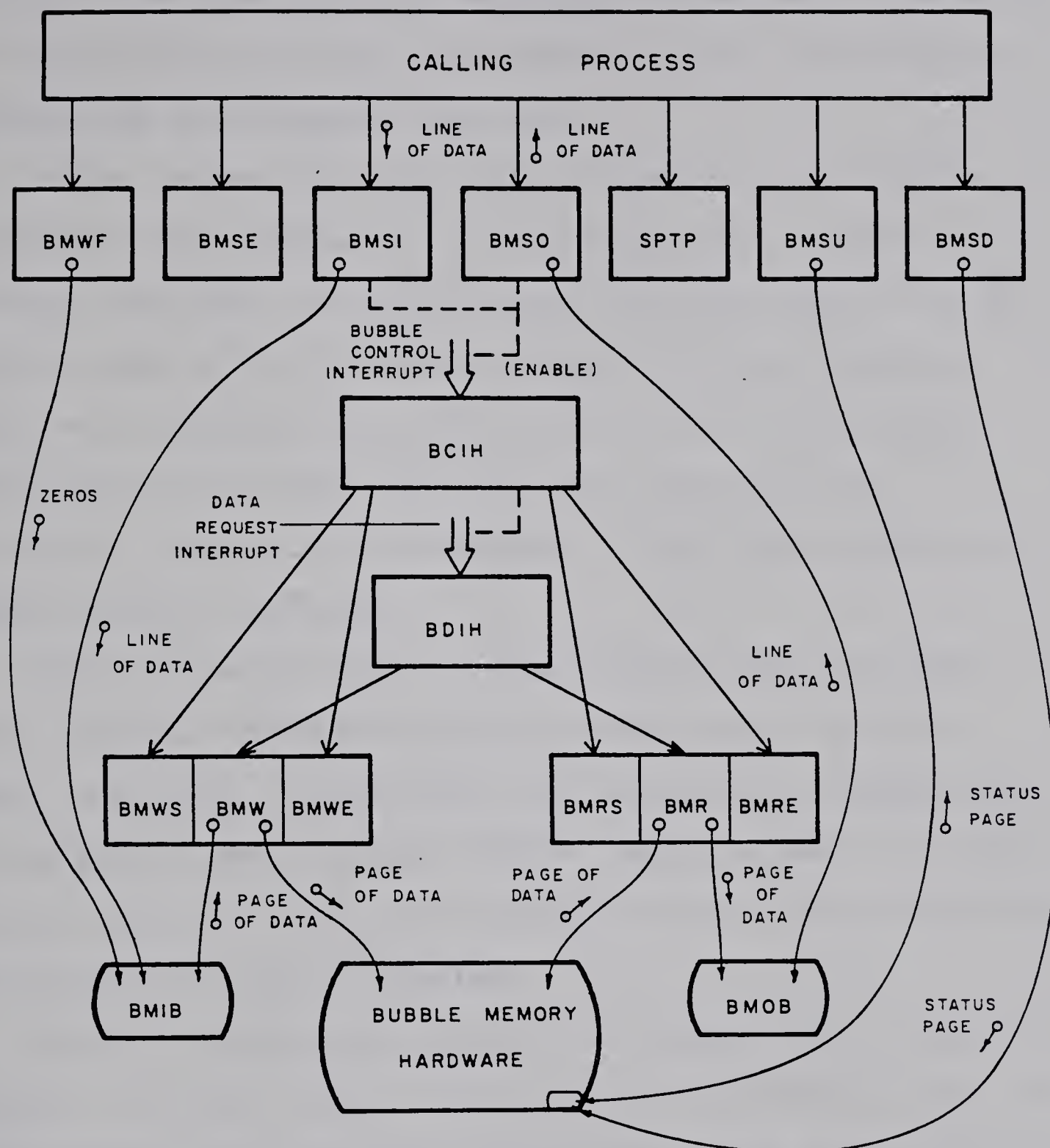


Figure IV.8 Structure of the Bubble Memory System Software

Whenever the bubble controller chip requires more data, a data request interrupt occurs which activates the data request interrupt handler, BDIH. BDIH will see that a write process is active and will call BMW which will transfer another ten bytes of data to the bubble controller chip.

This will continue until 64 data bytes have been transferred and the writing of a page is complete, after which BDIH will disable the data request interrupt.

After the writing of a page is complete, the bubble controller chip becomes idle which generates a bubble control interrupt. BCIH then senses that the page write is complete and calls the bubble memory write end routine, BMWE, which updates the status of the data in the bubble memory device. If BCIH then sees that there are no additional read or write requests, it will then disable the bubble control interrupt.

When a line of data is to be removed from the bubble memory system, the bubble memory system output routine, BMSO, is called. If BMSO sees that there is not enough data in the bubble memory output buffer, BMOB, to make up a line, a read request is made and the bubble control interrupt from the controller chip is enabled.

When the bubble controller chip becomes idle, a bubble control interrupt occurs. This activates BCIH which sees the read request and calls the bubble memory read start routine, BMRS. BMRS starts the reading of a page of data from the bubble memory system by sending a read instruction along with the address of the page to be read. The data request interrupt from the controller chip is then enabled.

When the bubble controller chip has read enough data from the bubble memory device, a data request interrupt occurs which activates BDIH. BDIH will see that a read

process is active and will call BMR which will transfer ten bytes from the controller chip to BMOB. This continues until the transfer of 64 bytes is complete and 14 bytes remain in the controller's small buffer. These 14 bytes are insufficient to cause a data request interrupt, but as the reading of a page is complete, the bubble controller chip becomes idle and a bubble control interrupt is generated.

Upon the bubble control interrupt, BCIH senses that the reading of a page is complete and calls the bubble memory read end routine, BMRE. This routine transfers the last 14 bytes from the controller chip to the BMOB and updates the status of the data in the bubble memory device. If BCIH then sees that there are no additional read or write requests, it will then disable the bubble control interrupt and the data request interrupt.

During or after the transfer of a page of data to the bubble memory hardware, if BCIH senses that the bubble controller chip has detected an error, the page read or write operation is restarted and the error is logged. After the successful completion of a page transfer, if BCIH detects a page read or write request, a new page transfer is started.

The bubble memory system start up and shut down routines, BMSU and BMSD, are used to recall and store the status of the bubble memory system in the bubble memory device itself. This status consists of the address of the next page to be written, the address of the next page to be

read, a flag to indicate whether the bubble memory system is empty, not empty, or filled to capacity, and a count of all the possible bubble memory system errors. This status is always stored at a fixed location in the bubble memory device called the status page. In addition to recalling the bubble memory status, BMSU also initializes the bubble memory system after power up by sending the appropriate instructions and information to the controller chip. BMSU is executed automatically when the microcomputer's power is turned on. BMSD is executed when the operator of the data acquisition and transmission system requests, just before the power is turned off, that data in the bubble memory system is to be retained.

The bubble memory write finish routine, BMWF, may be called to complete any partially filled page in BMIB and then start the write page process so that any data remaining in BMIB may be stored in the bubble memory device. The save partially transmitted page routine, SPTP, is used to alter the bubble memory system status to effectively save within the bubble memory device any data that has been read into BMOB but not yet transmitted by the transmission process. The bubble memory system empty routine, BMSE, may be called to set the status of the bubble memory system to indicate that there is no data in the bubble memory system, effectively emptying it.

E. Control

As described earlier, the operator of the data acquisition and transmission system can control this system through a set of keys on the microcomputer's front panel. The operator can monitor the operation of this system using the display mounted on the front panel of the microcomputer and the terminal.

In terms of operation, the data acquisition and transmission system can be in one of four states, as shown by the system control state diagram in Figure IV.9. At any one time the system may be idle, acquiring data, transmitting data, or both acquiring data and transmitting data. The change from one state to another occurs when the request for the appropriate control function is granted or if certain conditions occur within the data acquisition and transmission system. The system operator makes such a request by depressing a control function key on the front panel. If that particular function is allowed, the request is granted causing the execution of the control function and possibly resulting in a change in the system's state. Conditions within the system that will produce a change of state are when the bubble memory becomes filled to capacity causing data acquisition to stop and when it takes longer than five minutes to transmit a line of data causing data transmission to stop.

acknowledgement is eventually displayed. In the state diagram, a point before the control function table indicates that it is a second function associated with a function key on the front panel.

The SS (start sampling) function sets the filter output to zero, sets the flag or marker byte to "F0" to mark the first sample after sampling was started, and then starts the data acquisition process by enabling a 40 hertz interrupt from a programmable timer chip. This function is not executed if the SEL function was not executed already, if the bubble memory device is filled to capacity, or if the data acquisition process is already active. The acknowledgement that this function was executed is a flashing "S" in the display.

The .HS (halt sampling) function stops the data acquisition process by disabling the 40 hertz timer interrupt. This function will not be executed if the data acquisition process is not active. The execution of this function is acknowledged when the "S" stops flashing in the display.

The TX (transmit) function starts the process of sampled data transmission by activating the main line transmission driver routine. This function will be executed only if the SEL function has already been executed and the transmission process is not already active. The execution of this function is acknowledged by the appearance of "SEndinG" in the display and "\$COPY *SOURCE* TO R(*L+1)" or "R PROCESS

6=R(*L+1)" on the terminal.

The .noTx (no transmission) function indicates to the transmission driver that the process of sampled data transmission is to stop. This function is not executed if sampled data transmission is not active. The execution of this function is acknowledged when no new lines except "END OF FILE" appear on the terminal.

The SEL (select) function is used to specify which set of digital filter coefficients are to be used, which resampling frequency is to be used, and which mechanism, a copy command or a program, is to be used by the MTS system to receive data and place it in a file. This function can be executed only while the data acquisition and transmission system is idle. Details on the use and the acknowledgment of this function and the remaining functions are given in the operator's manual in Appendix A.

The .END function is used normally when the user wishes to stop the acquisition of data. This function halts the data acquisition process and then allows the transmission of data to continue until all the sampled data stored in the bubble memory system is transmitted. When the bubble memory system becomes empty, the transmission process is stopped. This function is executed only if the transmission process is active.

The CAL (calibrate) function is used for entering calibration points as part of the sampled data so that after transmission the routines which analyse or read this data

can calibrate it. Each input channel can be calibrated at two points to allow for a change in gain and an offset so that the data may represent the original activity monitored in the appropriate units. Each calibration point entry will appear as a line of sampled data which is automatically marked by a flag indicating it is to be used for calibration information. The flags "C1" to "C8" are used to indicate the first calibration points of channels 1 to 8, while the flags "D1" to "D8" are used to mark the second calibration points. The original signal levels, which these points represent, can be specified when the routines to read the sampled data from a file are executed. The calibration function cannot be executed when the data acquisition process is active.

The Flag function is used to set a flag or marker in the next line of sampled data to indicate a special point in time or some event or stage in the acquisition of data. It is not meaningful to execute this function if the data acquisition process is not active. The flags "F0", "C1" to "C8", and "D1" to "D8" cannot be entered.

The .TS (time set) function is used to set the 24 hour clock used by the data acquisition process to determine the time when each sampled data line is generated. This function cannot be executed while the transmission process is active as the same clock is used to determine how long it takes to transmit a line of data.

The .SIG (signon) function is used to signon to the MTS system. This function is normally the first one requested

after power up and can be executed only if the data acquisition and transmission system is idle.

The SAVE function is used to save the status of the bubble memory system in the bubble memory system status page. This allows any untransmitted data within the data acquisition and transmission system to be retained after the power is turned off and to be available for transmission after the power is turned on again. This function will be executed only if the data acquisition and transmission system is idle.

The .CLB (clear bubble system) function is used to set the status of the bubble memory system to indicate that there is no data in this system. Effectively all data is removed from this system. This function can only be executed when the data acquisition and transmission system is idle.

F. Other Devices

The microcomputer system monitor is interrupt driven from a programmable keyboard display interface, the Intel 8279 PKDI. The monitor is the lowest priority interrupt driven software module in the microcomputer system. The PKDI drives eight seven segment light emitting diode character displays and scans and debounces a keypad mounted on the front panel. A small random access memory in the PKDI holds the character pattern to be displayed by the light emitting diodes, while a small first in first out buffer maintains a record of unprocessed key closures.

When a key closure is detected, the PKDI generates an interrupt which will be serviced by the monitor. If a numeric key is depressed, the appropriate numeral is written into the display and the execution of the interrupted routine resumes. If one of the control function keys described in the previous section is depressed, the appropriate module to process a control function request is executed and, independently of whether or not the control function itself is executed, eventually the execution of the interrupted programming will resume. If one of the monitor system development function keys is depressed, the appropriate function is executed providing that section of the monitor has been enabled by a switch mounted on the microcomputer's chassis. Execution of the program interrupted by a system development function will usually not resume. By disabling this section of the monitor, accidental termination of the data acquisition or transmission processes can be avoided.

As the system development section of the monitor is interrupt driven, the execution of a program can be monitored or controlled by accessing the machine state ¹³ which is placed on the computer system's stack when an interrupt request is granted. This allows the contents of any of the processor's registers, at the point where a

¹³ The machine state has been defined as the contents of all the processor's registers.

G. Jack Lipovski, *Microcomputer Interfacing*, Lexington Books, D.C. Heath & Company, Lexington, 1980, p. 9

program was interrupted, to be displayed or altered by the execution of the appropriate function. The breakpoint function and the function that allows the execution of single program steps are implemented using an address trap. When a valid address appears on the address bus which is the same as that stored in a 16 bit latch, a nonmaskable interrupt is generated which stops the execution of the program and causes the execution of one of the monitor development functions. A more detailed description of these functions is contained in the microcomputer software listing in Appendix B, ROM2.

The test routines have been written as main line programs, that is, they are not written as subroutines. Some of these routines have been written so that after a successful execution they start executing again. Under these conditions, a particular test can be run indefinitely. When a test routine detects a failure or an error, it displays an error message and usually stops executing. These routines are usually stored in the same section of read only memory as the routines they test and are described more fully in the software listing in Appendix B.

V. Performance

In this section the performance of the final implementation of the data acquisition and transmission system is described. After implementation, if the performance of a particular design does not fulfill its requirements, one must try an alternate implementation or decide on a new set of specifications. If the requirements of a design cannot be met by alternate implementations or changes in the specifications, then the requirements must be relaxed.

A. Microcomputer System

All hardware components in the microcomputer system have been tested directly or indirectly and in the final design, no malfunctions have been detected. Wherever possible, hardware components have been tested before being used within the system. During system development, some causes of hardware problems were incorrect wiring, one component was incorrectly inserted into its socket, and others required sufficient power supply decoupling. The Intel 8279 programmable keyboard display interface and the Intel 7220-1 bubble memory controller chip behaved erratically after transients on the A.C. power lines. This problem was solved by providing additional power supply decoupling right across the appropriate pins of these chips and by providing A.C. powerline conditioning. No errors have been detected in the operation of any individual hardware

component in the system, including the processor, when implemented properly.

All software was tested thoroughly during and after its development and was carefully modified until it performed its function as required. As software was well structured, software changes were easy to make.

B. Data Acquisition Process

Initial Sampling

The programmable timer that generates the 40 hertz interrupt to drive the data acquisition process also generates a 1 hertz signal to drive the 24 hour clock used by the data acquisition system. This clock loses about 1 second in 8 hours which means that the interrupt request frequency is 40 hertz less .003 % . A maximum latency time of 1.3 milliseconds for this interrupt would occur if it coincided with the start of a bubble memory system page write and a transmission interrupt request. Under these conditions, before the sampling routine will have completed its execution, a bubble control interrupt, two bubble data request interrupts, and three transmission interrupts may occur. Although this event rarely occurs, it means that each initial data sample is taken within an 8 % range of the sampling period. This error as well as its rate of occurrence can be reduced by halting the transmission process.

In the data acquisition circuitry, the most serious cross talk occurs between adjacent channels because the

sample and hold device has a fixed slew rate (dependent on the holding capacitor). This cross talk was measured under worst case conditions for all 8 input channels. All inputs were set a 2.3 volts except one channel which was allowed to float causing the associated operational amplifier output to go to about -15 volts. The measured crosstalk was equivalent to .02 % of the 5 volt input range which is the magnitude of the analog to digital converter's least significant bit.

The data acquisition circuitry provides a resolution of 1.2 millivolts which is the equivalent input magnitude of the analog to digital converter's least significant bit. When the calibration feature of the data acquisition system is used, voltages are recorded within .1 % of full scale.

When all inputs are grounded, noise levels are at the most the magnitude of the least significant bit of the analog to digital converter and are attenuated by the digital filter.

Digital Filter

All sets of digital filter coefficients provide low pass characteristics with the appropriate cut off frequencies. When used in the digital filter implementation, these coefficients provide better than 50 dB attenuation of the Nyquist frequency, 20 hertz, and the A.C. power line frequency, 60 hertz, which folds onto the Nyquist frequency. In response to a step input, any one of the sets of coefficients used in the filter produces an 11 % overshoot and a 2 % settling time of seven resampling periods.

C. Transmission Process

The rate at which sampled data can be transmitted from the data acquisition and transmission system to the University of Alberta's Amdahl 470V/8 based computer system, operating under MTS, depends on the loading of both systems. The maximum possible rate of data transmission to the MTS system increases as the loading on that system is reduced. The microcomputer based data acquisition and transmission system can accommodate a higher transmission rate of data stored in the bubble memory system if the data acquisition process is inactive. Lines of sampled data can be transmitted at a rate of over four lines a second when both systems are heavily loaded and up to a rate of over ten lines a second when both systems are lightly loaded.

As explained earlier, if the rate of data acquisition is greater than the average rate of data transmission, the maximum period of time data can be acquired, without the loss of data, depends on the difference between these two rates and the size of the data buffer. If the resampling rate is two hertz or four hertz, the rate of data acquisition is less than the average rate of data transmission. As long as the Amdahl system remains up and as long as the transmission process remains active, data acquisition may continue indefinitely while at the most only a few lines of data will accumulate in the bubble memory system. If the resampling rate is eight hertz and the transmission process remains active, depending on the

loading on the Amdahl system, the data acquisition process may continue for at least 25 minutes and up to two or three hours before data acquisition must be halted. If the resampling rate is ten hertz, data acquisition may continue for at least 14 minutes and up to an hour and a half. If the data acquisition process is periodically halted, say to avoid the acquisition of meaningless or redundant data, the requirements imposed on the transmission process and the data buffer will be greatly reduced.

As described earlier, the process in the MTS system which receives the transmitted data and places it in a disk file may be an MTS copy process or a Fortran program. Both processes appear to provide about the same rate of data transmission, although the use of Fortran is about twice as expensive. When the Fortran program is active, any type of data or comments may be sent back to the terminal without affecting the transmission process, other than introducing a delay.

Whether the data acquisition process is active or not, transmission can be started or stopped any number of times without affecting the acquisition process or losing any data in transmission.

In the transmission of about one line in 20,000 it appears the "DC1" character is lost in the transmission from the front end of the Amdahl computer system to the microcomputer system. This seems to be the result of an excessive receive data interrupt latency time associated

with the transmission process. When the "DC1" character is lost, the transmission process is stopped and this is detected by the transmission process driver. One minute after a line of data has been transmitted, if still no "DC1" character is received, the transmission process driver routine will send a null line, (a carriage return character), to the front end and transmission then resumes. When such a situation occurs, no sampled data is lost in transmission but a minute's worth of data must be allowed to accumulate in the bubble memory system.

During the time the data acquisition and transmission system was being tested and developed, the only time any data was lost or altered was when the transmission of this data coincided with A.C. power line surges or transients. The loss of this data was due to the introduction of parity errors by the effects of the powerline transients on the electronics within the modem or the terminal. This problem can be overcome by the use of an A.C. powerline conditioning device.

D. Bubble Memory System

When implemented with a 7110 one megabit device, the bubble memory system can hold up to 5453 lines of sampled data. At a two hertz resampling frequency this is equivalent to 45 minutes of data, while at a ten hertz resampling frequency this is equivalent to nine minutes of data. Data read or write errors in the bubble memory system have not

been detected during system development, testing, or use.

Data can be transferred to or from the bubble memory system faster if it is transferred in only one direction at a time. This is because the bubble memory access time is greatly reduced when consecutive pages are accessed which would be the case if, say, a number of pages were read from the system.

E. Control

The data acquisition and transmission system's control functions make operation of the system quite easy. Any control function that may unexpectedly interfere with one of the processes in that system cannot be executed when that process is active.

F. Other Devices

The monitor is sufficient for running and finding errors in most routines or programs. There are a few problems, though, when it is used to step through interrupt driven routines or programs. The test routines appear to be sufficient in testing all the devices and processes within the microcomputer system.

G. System Timing

Within the data acquisition and transmission system, although a number of processes are said to operate concurrently, the single processor in the system can be

executing only one process at any one time. All the processes within the system compete for the processor's time through the interrupt control mechanisms, and if for some reason, the processor does not have enough time, the execution of some processes will not be completed. The execution times of critical routines within the system are given in Table V.1.

Table V.1 Routine Execution Times

ROUTINE	EXECUTION TIME in milliseconds
SAMP	.9
FILT	14.5
RESAMPLING i)	1.0
ii)	2.0
BMSI	1.680
BMWS	.741
BMW	.438
BMWE	.048
BMSO	1.268
BMRS	.151
BMR	.453
BMRE	.694
TX2	.095
RX2	.056

As the sampling and filter routines are executed every 25 millisecond sampling period, (a rate of 40 hertz), and since their execution requires 15.4 milliseconds, this

leaves 9.6 milliseconds out of every 25 milliseconds for the execution of all other processes implemented within the microcomputer system. The resampling process will consume about one millisecond of the processor's time on the resample preparing a line of sampled data from the filter output, and will consume about two milliseconds in one of the next sampling periods storing this line in the bubble memory system.

The timing of the transmission process driver is not critical as this process has the lowest priority in the microcomputer system and executes only when the processor has "spare" time. But the transmission of a line of data obtained by this process is given the highest priority in the system due to the priority given to the ACIA interrupts. As the rate of data transmission is 9600 baud, and as the TX2 routine requires about 95 microseconds to transmit a character, at the most 2.5 milliseconds out of a sampling period may be used in data transmission. (The receive section of the ACIA is mostly idle.)

The timing of the reading and writing of a page within the bubble memory system is shown in Figure V.1. The routines BMSI and BMSO allow interrupts for the most part, and in turn activate their subordinate routines which do not allow interrupts while they are executing. The figure shows how execution of the bubble memory system routines proceeds if BMSO is called to request the reading of a page while the process of writing a page of data within the bubble memory

system is already underway. Shaded areas indicate the time interrupts are not allowed.

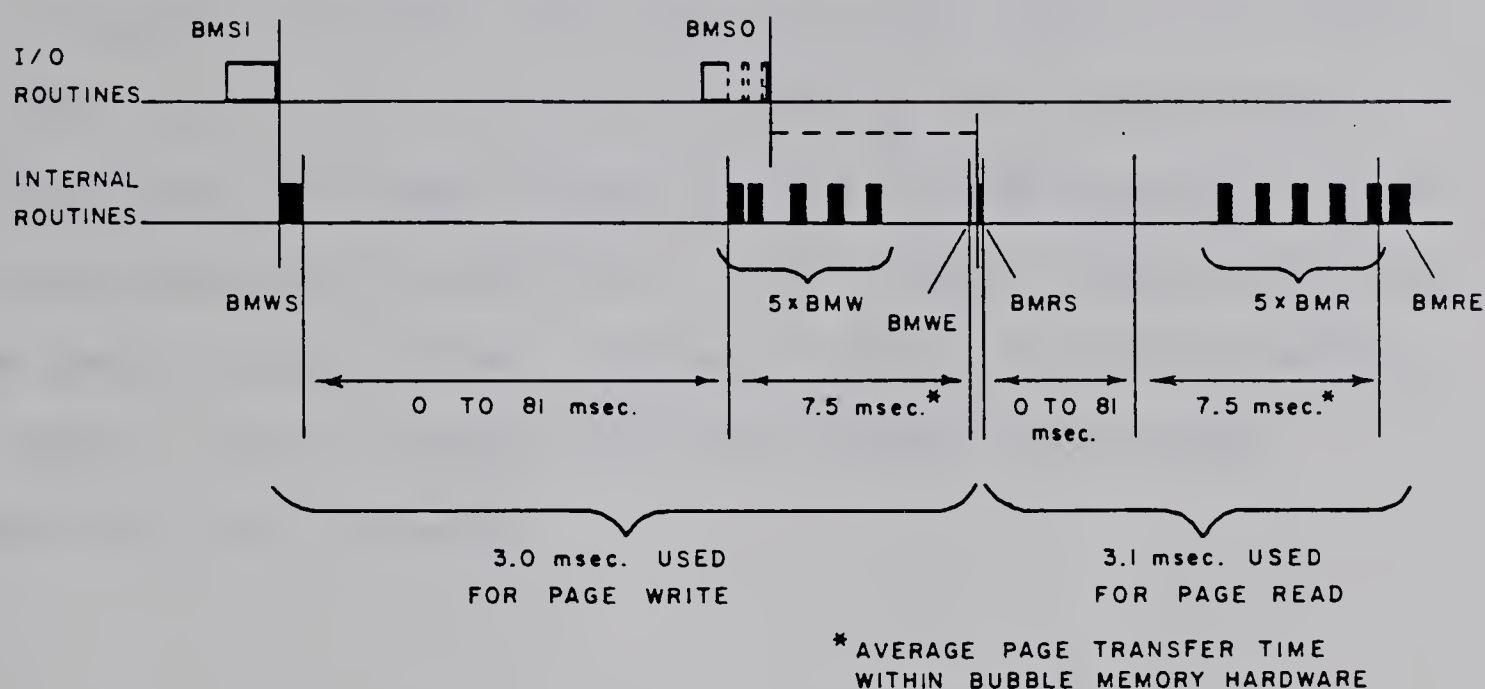


Figure V.1 Read and Write Page Timing

When BMSI is called, as the bubble controller chip is idle, the page write process is immediately started and 14 bytes are transferred to the controller. After the address of the page to be written is accessed in the bubble memory device, the transfer of 64 bytes from the controller chip to the other bubble memory hardware will take place in about 7.5 milliseconds.¹⁴ Although the BMSO routine may be called while the page write process is still active, the bubble memory read routines do not start execution until the page

¹⁴The average bit transfer rate from the controller chip to the formatter and sense amplifier chip is 68 kilobits/sec. BPK 72 Bubble Memory Prototype Kit User's Manual, Intel Corporation, Santa Clara, 1980, Order Number 121685-002, p. 1-5

write process is successfully completed. After the page read has been started and the page to be read is accessed in the bubble memory device, 64 bytes will be transferred to the controller in about 7.5 milliseconds.

Figure V.2 shows the processor time used by interrupt driven routines during two consecutive sampling periods. During the first sampling period the transmission of a line of sampled data is completed. In the second sampling period, the sampling and filter routines require 15.4 milliseconds, as usual, and it happens that the resampling process requires 2 milliseconds.

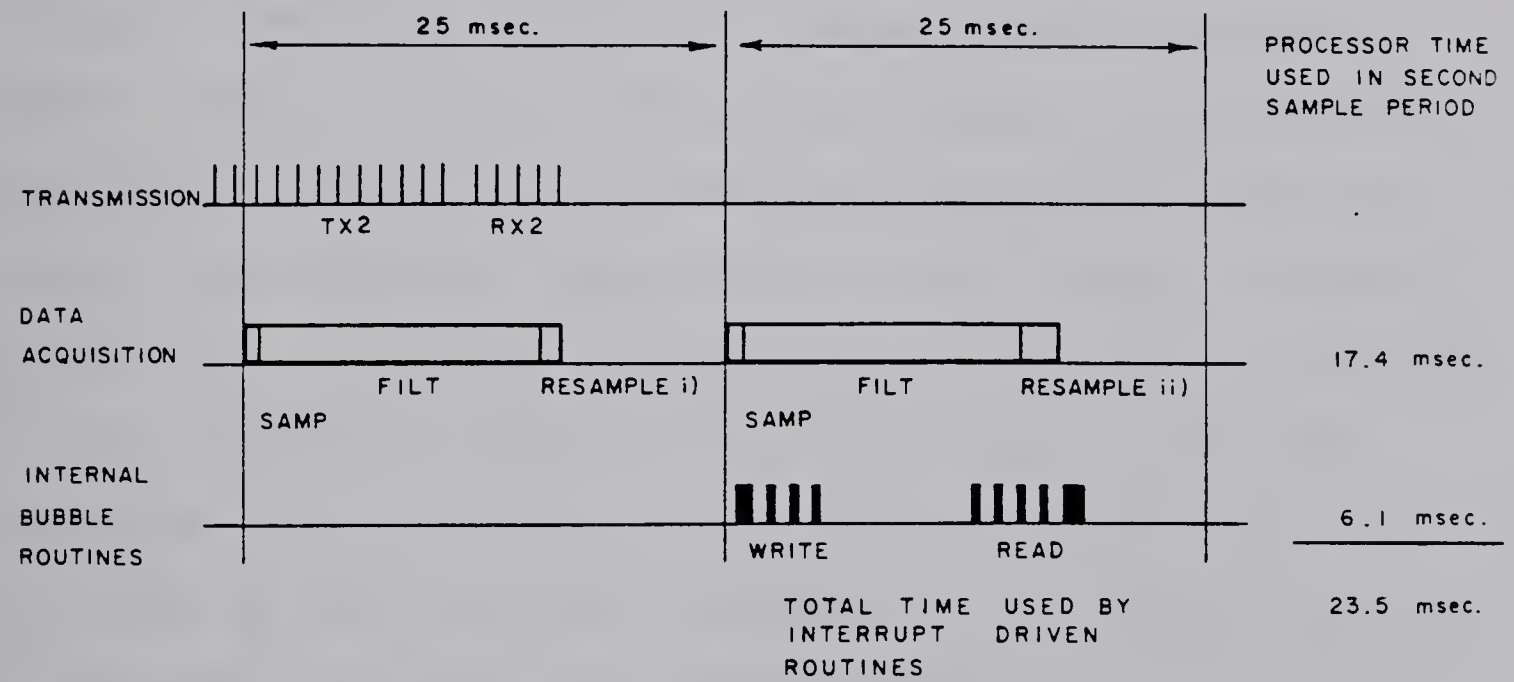


Figure V.2 System Timing

There is also a transfer of two pages ¹⁵ of data in the bubble memory system consuming 6.1 milliseconds. Three page transfers will not occur within one sampling period as at least 8 sampling periods are required to gather enough data to generate a page write request. ¹⁶ Transmission of data is not concurrent with the reading of a page of data as the transmission process driver waits until a line (or page) is transmitted before obtaining the next line (or page).

Although the chance of the above situation occurring is small, the processor in the second 25 millisecond period executes 23.5 milliseconds of interrupt driven code. This leaves 1.5 milliseconds before the next sampling period begins. An extra 2.0 milliseconds of execution time can be obtained, if necessary, by delaying the execution of the resampling code until the latter part of the next sampling period. This is done by using a global variable to indicate if the resampling code should be executing. If for some reason up to 3.5 milliseconds of extra execution time (of interrupt driven routines) is needed in the second sampling period, then the execution of the resampling code would be interrupted by the data acquisition process. As the status of the resampling code is saved in a global variable, the

¹⁵ Within a sampling period there may be a page read and a page write.

¹⁶ Each resampling generates 24 bytes of data. There are 64 bytes of data in a page. A resample is obtained, at the most, every fourth sampling period. If 16 bytes remain in BMIB after the previous page write, then after 2 resamples, a new page write request will be generated.

execution of this code may resume after execution of the digital filter algorithm in the next one or two sampling periods.

It is clear that the resources of the microprocessor, mainly its execution time, are being taxed to the limit. For the two hertz resampling rate, at least 40 sampling periods separate any two page write requests. As data can be transmitted only as fast as it can be acquired, at least 40 sampling periods must pass between page reads and the transmission driver is mostly waiting. On the other hand, for the ten hertz resampling rate, much more processor time is spent transmitting data and transferring data within the bubble memory system. It would seem that concurrent data acquisition, at a ten hertz resampling rate, and sampled data transmission would be the maximum load the data acquisition and transmission system can handle, although a 13.3 hertz resampling rate may be possible.

VI. Applications

As explained earlier, the function of the data acquisition and transmission system is to sample and to transmit to the University's Amdahl computer system data from low frequency signals of biological origin. This is to be done in such a way so that the frequency spectra of the original signals may be analysed. To demonstrate this function, electrical activity and movements in the wall of the transverse colon of a dog were made in vivo. The changes in the frequency spectra obtained versus time are plotted in Figure VI.1 and Figure VI.2. One can see a dominant frequency component at 4 cycles per minute. As a comparison, Figure VI.3 shows the same analysis of an artificially generated triangular wave that slowly ranges from about 1 cycle per minute to 6 cycles per minute.

Besides acquiring data from low frequency biological signals, the data acquisition and transmission system can be used to sample and transmit any signals which can be conditioned to have a maximum frequency component of 20 hertz, a maximum signal range between plus and minus 2.5 volts, and where the highest frequency component of interest is under 4 hertz. This data may be transmitted to any computer system which will accept ASCII characters through an asynchronous serial communications port and which can implement the handshaking protocol described using the ASCII "DC1" control character.

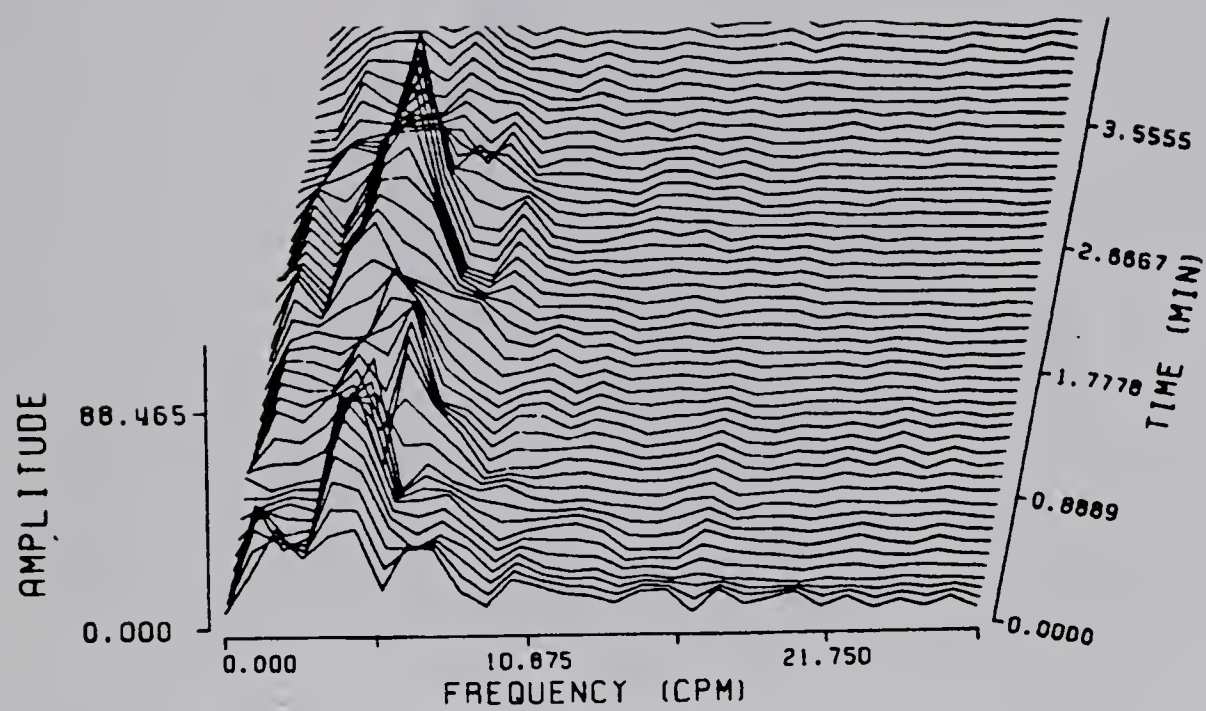


Figure VI.1 Analysis of Electrical Activity

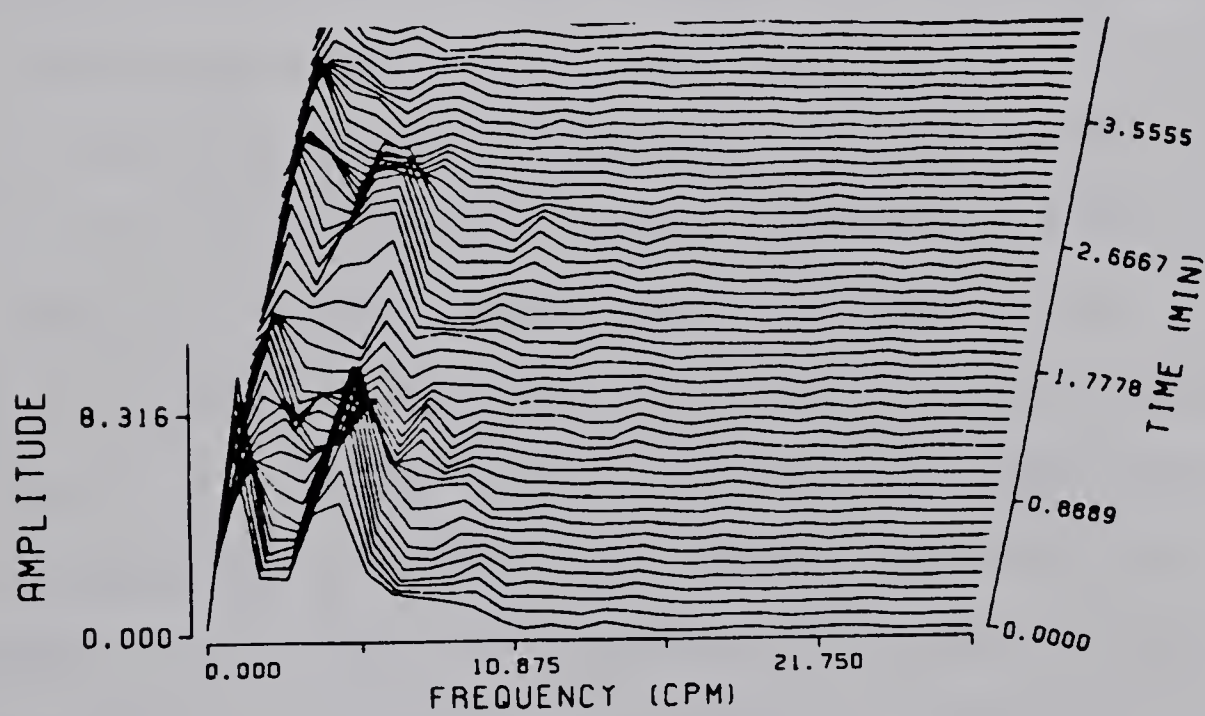


Figure VI.2 Analysis of Changes in Displacement

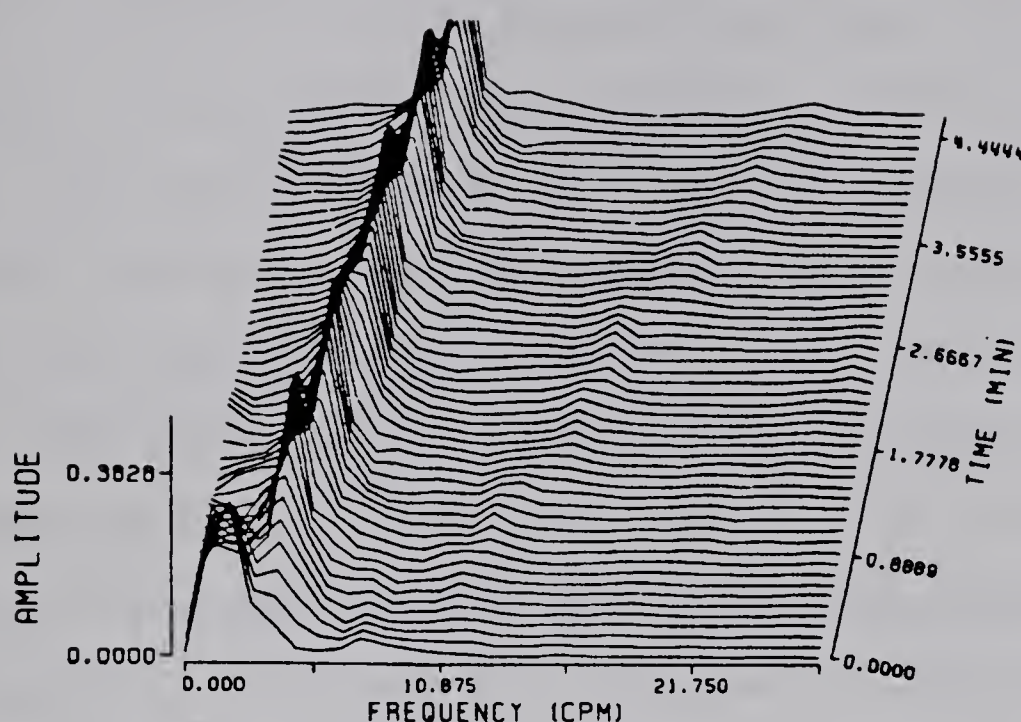


Figure VI.3 Analysis of Triangular Wave

Although a selection of four resampling frequencies is provided, each associated with a particular filter cut off frequency, other cut off frequencies and lower resampling frequencies can be provided by simply extending a table in the microcomputer system source code. This would give the system more flexibility for sampling slowly varying signals.

If the digital filter is not required, through minor software changes and a larger bubble memory system page size, higher initial sampling frequencies can be provided. Even higher initial sampling rates may be provided if the transmission process is idle while the data acquisition process is active.

As developed, the data acquisition and transmission system has enough flexibility to allow data acquisition to

proceed concurrently with the analysis of data already transmitted to the MTS system. The analysis would be done by a program operating under MTS and would be controlled in one of two ways. One way to achieve this analysis would be to run a program that reads the data transmitted by the data acquisition and transmission system and places this data in a disk file. When this program detects a particular flag in a line of sampled data, it will do an analysis of the data it has already received. Another way to achieve this would be to use the MTS \$COPY command to receive the data and place it in a file. When the user wishes to analyse any data already transmitted, he can halt the transmission process and run a program to perform the data analysis. In either case, while the analysis is being performed, data transmission will be delayed and acquired data will accumulate in the bubble memory system. The results of such an analysis may be sent back over the original line to be output on the terminal, or a device in series with it, or may be sent via another line to an output device such as a graphics plotter.

VII. Refinements and Improvements

Although the data acquisition and transmission system can perform its function reliably, changes can be made in the design to further improve its performance. Some of the possible changes mentioned in this section will be minor affecting only the implementation of certain devices or processes. Other changes would require alterations in the overall design of the system.

Minor system changes would include improving the digital filter, increasing the capacity of the bubble memory system, and implementing an automatic power up microcomputer system test. The characteristics of the digital filter can be improved, mainly the stop band attenuation, by using a 24 by 24 bit multiplier chip rather than a 16 by 16 bit chip. This would involve some changes to hardware and changes to two subroutines. The capacity of the bubble memory system can be doubled, quadrupled, or octupled by adding one, three, or seven more 7110 bubble memory devices and their associated interface circuits. Such a change would essentially be a hardware change, as only a few constants would have to be changed in the bubble memory system software package. A microcomputer system power up test could be implemented with only software changes in the power up reset procedure and the testing routines. The testing routines would have to be rewritten as a number of subroutines, each testing a particular device or process in the microcomputer system. Each routine would return control

to the reset procedure if the test was successful, or provide an error message and perhaps enter a monitor system development function if the test failed.

A change which would simplify the bubble memory system software, would allow for a greater page size, would greatly reduce the processor time spent servicing the bubble memory system, and would reduce bubble memory page read and write times, would be the use of a direct memory access (DMA) controller to transfer data between random access memory and the bubble memory controller chip. This added performance would be gained at the expense of requiring some extra hardware to implement DMA control.

One way to improve the overall performance of the data acquisition and transmission system would be to use a more powerful microprocessor. To use the features of a more powerful processor would at least require significant changes in the source code, if one can manage to avoid rewriting all the code. Changes in the structure of the software may even be required and depending on the processor chosen, hardware changes may also be required.

Finally the performance of the data acquisition and transmission system would be increased the most by an overall system design change. If a system with two microprocessors were developed, the first processor could handle data acquisition, including sampling, filtering, and resampling, while the second would control data buffering and data transmission.

In this design, the first microprocessor could be one of the more powerful processors to speed up the execution of the digital filter algorithm and thereby allow a greater system input bandwidth. The use of a hardware multiplier may still be advantageous. As this processor would only have to service an interrupt occurring at the rate of the initial sampling frequency, samples would be taken at intervals of precisely the selected sample period.

The second processor in this design would not have to be as powerful as its function would be one of control, rather than computation. This processor would take the resampled data from the first processor, store and retrieve the data in a bubble memory system with the aid of a DMA controller, and then, whenever possible, transmit the data to the larger computer system. This second processor would have plenty of time to interface to the system operator and even respond to or handle any information received from the main frame computer system.

Bibliography

Davis, J.E., "The 7110, a One Megabit Magnetic Bubble Memory", Reliability Report 22, Intel Corporation, Santa Clara, September 1979

Enos, J.C., Van Tilburg, R.L., "Tutorial Series - 5: Software Design", Computer, IEEE Computer Society, Vol. 14, No. 2, Feb. 1981

Intel Corporation, Intel Component Data Catalog, Intel Corporation, Santa Clara, 1980

Intel Magnetics Incorporated, BPK 72 Bubble Memory Prototype Kit User's Manual, Intel Corporation, Santa Clara, Order Number 121685-002, 1980

Jensen, R.W., "Tutorial Series - 6: Structured Programming", Computer, IEEE Computer Society, Vol. 14., No. 3, March 1981

Lam, H.Y-F., Analog and Digital Filters: Design and Realization, Prentice-Hall, Englewood Cliffs, 1979

Leventhal, L.A., Introduction to Microcomputers: Software, Hardware, Programming, Prentice-Hall, Englewood Cliffs, 1978

Leventhal, L.A., 6809 Assembly Language Programming, Osborne/McGraw-Hill, Berkeley, 1981

Lipovski, G.J., Microcomputer Interfacing, Lexington Books, D.C. Heath & Company, Lexington, 1980

The Microprocessor and Its Application: An Advanced Course, Aspinall, D. editor, Cambridge University Press, Cambridge, 1978

Motorola Incorporated, M68MM01A/1A2 Monoboard Microcomputer Micromodule 1A/1A2, Motorola Incorporated, Phoenix, May 1979

Motorola Technical Information Center, The Complete Motorola Microcomputer Data Library, Motorola Incorporated, Phoenix, 1978

Myers, G.J., **Composite/Structured Design**, Van Nostrand Reinhold, New York, 1978

Nagle, H.T. Jr., Nelson, V.P., "Digital Filter Implementation on 16-Bit Microcomputers", **Micro**, IEEE Computer Society, Vol. 1, No. 1, Feb. 1981

Noel, R., Engley, B., **The Michigan Terminal System Terminal User's Guide**, The University of Alberta Computing Services Department, Edmonton, 1978

TRW Incorporated, **TRW LSI Multipliers, HJ Series**, TRW Incorporated, Redondo Beach, 1978

Appendix A -- Operator's Manual

General Function

The function of the data acquisition and transmission system is to sample, digitize, and transmit data simultaneously from 8 low frequency analog signals. The system has a large nonvolatile data buffer to store data until it can be transmitted. All 8 inputs are filtered and then sampled at the same rate, but a 2, 4, 8, or 10 hertz rate may be selected. This sampled data is transmitted to the University of Alberta's Amdahl computer operating under the Michigan Terminal System, (MTS).

The data acquisition and transmission system can be controlled from its front panel, a representation of which is shown in Fig. 1. There is a switch to turn on the power and a light to indicate that power is on. There is a keyboard and a 7 segment LED display to be used by the system operator. The keys are divided into three groups; the first group is used for data entry only, the second group is used to control the data acquisition and transmission system, and the third group was used to develop the system and can be used to test the system. In normal operation the system development keys are disabled by a switch mounted on the chassis inside the system. The operation of these key is explained by the source code listing in Appendix B. The display is used to indicate what data has been entered, what the status is of an active control function, and the status of the data acquisition and transmission system. There is also a rotary switch used to connect the line from a modem to the system or to an external device such as a graphics terminal.

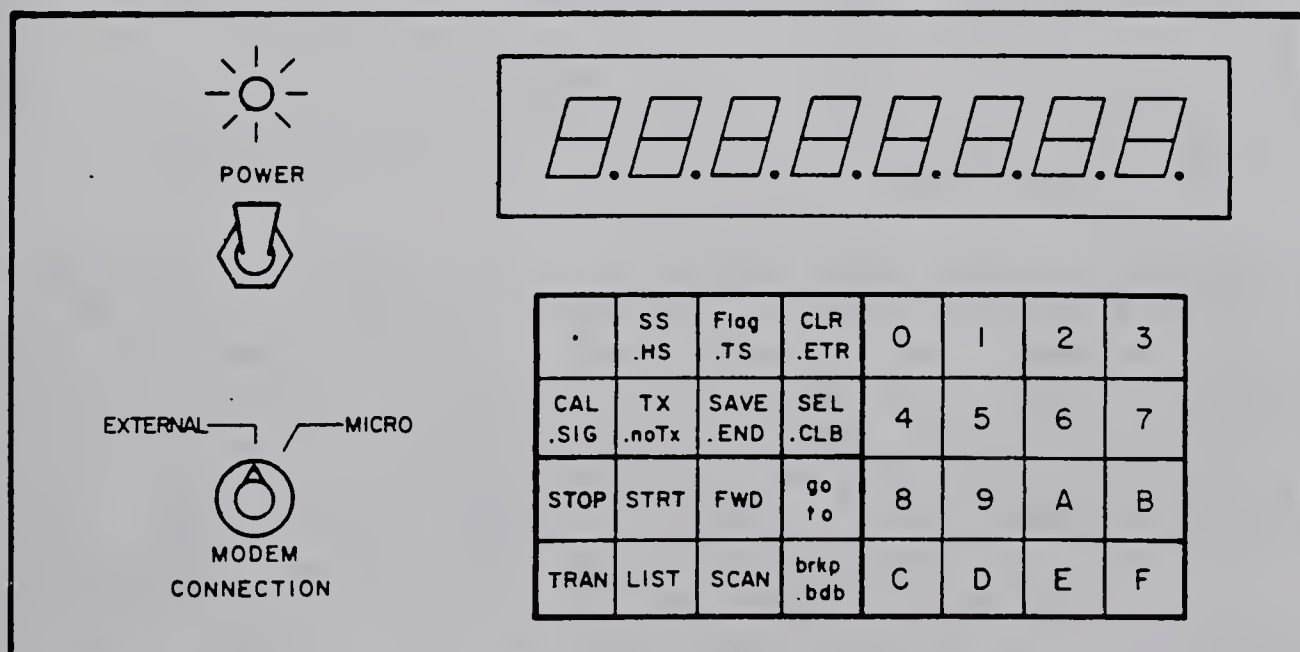


Fig. 1. FRONT PANEL

The 8 analog inputs to the data acquisition and transmission system enter through an RS232 type plug mounted on the back panel represented in Fig. 2. The pinouts for

this connection are shown in Fig. 3. These 8 inputs are not isolated, but are buffered through operational amplifiers. The signals on these inputs should remain within ± 2.5 volts of the A.C. powerline ground and must remain within ± 15 volts. The signals must contain no frequency components above 20 hertz if they are to be accurately represented by the sampled data. Transmission occurs through an RS232 standard asynchronous serial communications port mounted as an RS232 plug on the back panel. 7 bit ASCII code is transmitted at 9600 baud with even parity and one stop bit. This RS232 port, marked "MODEM", should be connected to a dumb terminal and the terminal should be connected to a line from the computer via a modem. The terminal will allow transmission to be monitored and will allow descriptive data to be entered. Also mounted on the back panel is an RS232 plug marked "EXTERNAL" which can be used to connect an external device to the modem via the rotary switch mounted on the front panel.

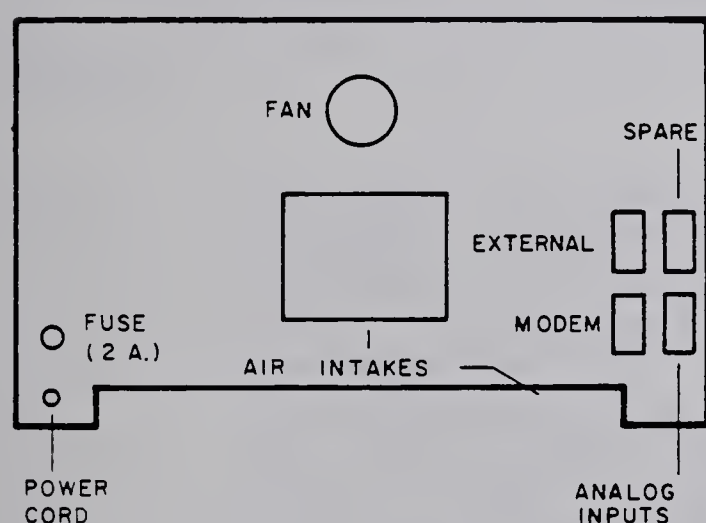


Fig. 2. BACK PANEL

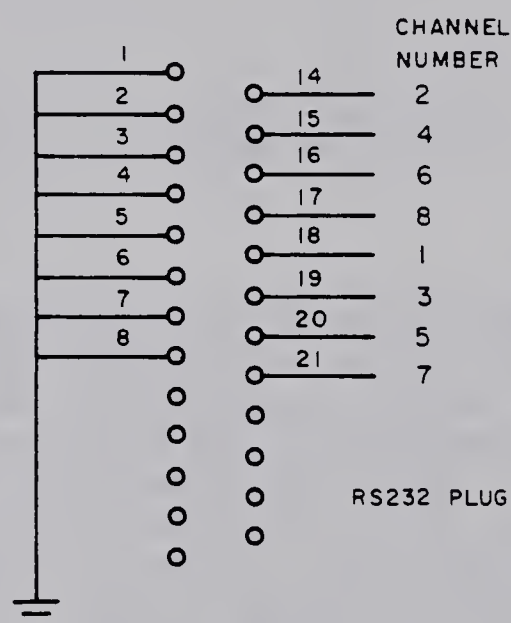


Fig. 3. ANALOG INPUT PINS

As data is acquired at a fixed rate and is transmitted to a time shared computer system, a bubble memory system is used to implement the required data buffer. Depending on the rate of data acquisition and the loading of the University's computer, data acquisition may proceed uninterrupted for a certain length of time before the data buffer is filled to capacity. The values shown in Table 1 have been determined assuming that the data buffer is empty at the start of the data acquisition process. As the data buffer is a nonvolatile memory device, data may be retained while the power is tuned off.

Table 1. MAXIMUM PERIOD OF UNINTERRUPTED
DATA ACQUISITION

SAMPLING RATE (Hz.)	2	4	8	10
NO TRANSMISSION	45.4 min.	22.7 min.	11.4 min.	9.1 min.
LOADING OF HEAVY AMDAHL COMPUTER LIGHT	INDEFINITE	INDEFINITE	OVER 22 min.	OVER 14 min.
	INDEFINITE	INDEFINITE	2 to 3 hr.	1.5 hr.

Function Keys

The following is a description of the function keys used to operate the data acquisition and transmission system.

- (.) This key is to be depressed before a function key in order to select the second function associated with the function key. On a function key, the label for the first function is always printed on the upper half of the key. The label for the second function is printed on the lower half and the label always begins with a point. The first time this key is depressed a point appears in the display indicating a second function request has been made. If this key is depressed a second time, the point disappears and the second function request is cancelled.
- (CLR) Clear. This function is used to clear the display. It also clears the data registers described next. It can also be used to exit the system control functions that request the entry of data, namely (CAL), (SEL), and (.CLB).
- (.ETR) Enter. This function is used to enter data into two 16 bit data registers. These registers are used only by the system development functions and have no effect on the operation of the data acquisition and transmission system.

(.SIG) Signon. This function can be used to signon to the University's computer with the I.D. and password stored as ASCII code in locations BF7A and BF82 of ROM8. After the keys, (.),(.SIG), have been depressed, "SiG" is displayed and the signon procedure commences. The signon procedure can be watched from the terminal and when the signon is complete, "SiGnEd.on" is displayed. The signon procedure consists of a number of steps and each step is reexecuted until successfully completed. If it becomes apparent that this procedure is not working, the procedure can be halted by entering (.),(.SIG) and "SiG. FAIL" will appear. One may then restart the procedure by entering (.),(.SiG) again.

This function will not be executed if the data acquisition process, the transmission process, or another function is already active. In this case "Error" is displayed. "TrY.LATeR" is displayed if the user must only wait until the remaining data in the bubble memory is transmitted, (in this case he should be signed on already anyway).

(.TS) Time Set. This function is used to set the 24 hour clock associated with the data acquisition process. The time is entered in decimal form where the hours must be less than 23 and the minutes and seconds less than 59. The required keystrokes, enter the time 22:01:05, are as follows.

keystrokes:	display:
(CLR)	" "
(2),(2)	"22 "
(0),(1)	"2201 "
(0),(5)	"220105 "
(.),(.TS)	"220105TS"

After the above keystrokes, "TS" is displayed if the clock was successfully set, otherwise the display is cleared. "Error" is displayed if the user tries to execute this function while the transmission process or another function is active. "TrY.LATeR" is displayed if the user only has to wait until the remaining data in the bubble memory system is transmitted (and when transmission will stop automatically).

(.CLB) Clear Bubble. When this function is executed the

bubble memory system is effectively emptied. The required keystrokes are as follows.

keystrokes:	display:
(.),(.CLB)	"MT. B.S? "
(A),(1)	"B.S. MT "

After "MT. B.S?" is displayed, the bubble memory system will be emptied only if the confirmation "A1" is entered. When "B.S. MT" is displayed, the system has been emptied.

"Error" is displayed if the data acquisition process, the transmission process, or another function is active. "TrY.LATeR" is displayed if the transmission process will stop after the remaining data in the bubble memory is transmitted.

This function effectively empties the bubble memory system by setting the system's status flags to indicate that it is empty. No data in the bubble memory device is altered until the data acquisition process is started or until the (SAVE) function is used.

(SEL) Select. This function is used to specify the rate at which data is sampled, the cut off frequency of the filter, whether or not the bubble memory system is to be used, and the process used by MTS to receive data and place it in a file on disk. To execute this function for the first time after power up, the following keystrokes are used.

keystrokes:	display:
(SEL)	"SEL. "
(x),(y)	"SEL. 10 4"

The next time this function is executed, the following keystrokes are required.

keystrokes:	display:
-------------	----------

(SEL)	"rESEL?"	"
(A),(1)	"SEL."	"
(x),(y)	"SEL. 10 4"	

"rESEL?" indicates the user is to enter the confirmation, "A1", before a reselection is made. "SEL." indicates that the user is to enter the selection code, "xy" (two data keystrokes). "SEL. 10 4" indicates that the selection function has executed successfully. The effect of the selection code, "xy", is as follows.

Value of x :	Effect:
-----------------	---------

- | | |
|---|--|
| 0 | An MTS \$COPY command is used to transfer data to file R. The bubble memory system is used as a data buffer. The filter has a cut off frequency as specified by the second digit, "y". |
| 1 | The (compiled) program, PROCESS, is run under MTS to place data in a file. (Logical unit 5=*source*, 6=R, see Appendix C.) The bubble memory system is used as a data buffer. The filter has a cut off frequency as specified by "y". |
| B | The \$COPY command is used. The bubble memory system is used as a data buffer. The digital filter coefficients are set so the filter is effectively bypassed. |
| E | The \$COPY command is used. The bubble memory system is bypassed. The filter has a cut off frequency as specified by "y". This selection and the next are used to test the data acquisition process. If the terminal is set in the half duplex mode and RAM location 0021 is set to 0F with the system development functions, data will be displayed on the terminal without the use of a connection to MTS. |

F The \$COPY command is used. The bubble memory system is bypassed. The filter is bypassed as in selection "B".

Value Effect:
of y :

- | | |
|---|---|
| 2 | A two hertz sampling rate is selected. If not bypassed, the filter provides a cut off frequency of .6 hertz. |
| 4 | A four hertz sampling rate is selected. If not bypassed, the filter provides a cut off frequency of 1.225 hertz. |
| 8 | An eight hertz sampling rate is selected. If not bypassed, the filter provides a cut off frequency of 2.45 hertz. |
| A | A ten hertz sampling rate is selected. If not bypassed, the filter provides a cut off frequency of 3.0627 hertz. |

If a selection code other than one of the 20 possible combinations of the above is entered, "Error" is displayed. (The process to be used by MTS to receive data may have been changed!) "Error" is displayed if the data acquisition process, the transmission process, or another function are active. "TrY.LATeR" is displayed if the user only has to wait until remaining data in the bubble memory has been transmitted.

(TX) Transmit. This function is used to start the transmission process. The user simply depresses the (TX) key. "COPY *SOURCE* TO R(*L+1)" or "R PROCESS 6=R(*L+1)" will appear on the terminal if a connection has been made to MTS. When transmission is ready to begin, "SEndinG" is displayed. At this point, descriptive data may be entered into the file R. (See 'Typical Operation'.)

"USE SEL" is displayed if the user must first execute the (SEL) function. "Error" is displayed

if the transmission process is already actively transmitting data or if another function is active. "B.S. MT" is displayed when the transmission process is active but cannot transmit anything because the bubble memory system is empty. This function can be executed while the data acquisition process is active.

(.noTx) No Transmission. This function is used to stop the transmission of data to MTS. The required keystrokes are (.),(.noTx). When transmission is stopped, the terminal will beep and "END OF FILE" and "* TRANSMISSION STOPPED *" will appear on the terminal.

This function is effective only while data is actually being transmitted. If the transmission process is active but waiting for more data, (evident by no new lines of data appearing on the terminal or when (TX) produces "B.S. MT"), the (.END) function should be used. "Error" is displayed if the user tries to execute this function while another function is active.

(CAL) Calibrate. This function is used to enter calibration points into the sampled data so that acquired data may later be calibrated by programs running under MTS. (See the subroutines of program PRNTD in Appendix C.) The required keystrokes are as follows.

keystrokes:	display:
(CAL)	"CAL. "
(x),(y)	"CAL. 10 4"

"CAL." indicates that the user is to enter the calibration code, "xy" (two data keystrokes). After the calibration is complete, (wait a few seconds), "CAL. 10 4" is displayed.

For each channel, the user should enter two calibration points to allow for a change in gain and an offset. "C" is entered for "x" to specify that the first calibration point is to be entered for a channel, while "D" is entered for the second calibration point. "y" may be the digits "1" to

"8" to specify one of the eight channels.

For example, if channel 5 is used to monitor a signal derived from a pressure transducer, this channel can be calibrated at 0 cmH₂O and 100 cmH₂O. First the user subjects the pressure transducer to zero (gauge) pressure and then enters (CAL),(C),(5). He then should wait until "CAL. 10 4" appears in the display. To enter the second calibration level, the user subjects the transducer to 100 cmH₂O of pressure and enters (CAL),(D),(5). Later, when a program is run to analyse the data transmitted to MTS, the program will ask the user what the calibration levels were. (The channels calibrated and the levels they were calibrated at should be entered by the user into the data file as descriptive data.) NOTE: Adjustments should not be made on any equipment used to condition the input signal for a particular channel once that channel has been calibrated.

"Error" is displayed if the calibration code entered is not within the range "C1" to "C8" or "D1" to "D8". "Error" is also displayed if the user tries to execute (CAL) while another function is active. "USE .HS" is displayed if the user must use the (.HS) function first to stop the data acquisition process. "USE SEL" is displayed if the user must first use the (SEL) function.

(SS) Start Sampling. This function is used to start the data acquisition process. After the (SS) key is depressed, an "S" will flash in the display at one half the sampling frequency.

"Error" is displayed if another function is already active. "USE SEL" is displayed if the (SEL) function must be used first. "B.M.S.FULL" is displayed if the data acquisition process is not allowed to start because the bubble memory system is filled to capacity. Using this function while the data acquisition process is already halted has no effect. The first line of sampled data, taken after (SS) is used, is marked with the flag "F0".

(HS) Halt Sampling. This function is used to stop the data acquisition process. After the (.) and the (.HS) keys are depressed, the "S" will stop

flashing in the display indicating that sampling has stopped.

"Error" is displayed if another function is already active. Using this function after sampling has already stopped has no effect.

(Flag) Flag. This function is used to set a flag or marker byte in the next line of sampled data to indicate something special about that sample or the time it was taken. Flags may also be used to communicate something to a program. To enter the flag "45" the following keystrokes are used.

keystrokes:	display:
(CLR)	" S "
(4), (5)	"45 "
(Flag)	"4545 S "

The "S" continues to flash in the display as the execution of this function is meaningful only if the data acquisition process is active. "45" is displayed a second time to indicate that the flag has been properly entered, otherwise "00" would be displayed.

Flags which cannot be entered are "FF", "F0", "C1" to "C8", and "D1" to "D8". It is meaningless to enter the flag "00". "Error" is displayed if another function is active.

(.END) End. This function is normally used at the end of the data acquisition session. It stops the data acquisition process and allows the transmission of sampled data to continue, (if it is active), until the bubble memory system is empty. After this function is executed, once the bubble memory system is empty, the transmission process is automatically stopped. This function can also be used to stop transmission while that process is waiting for data to be entered into the bubble memory system, (ie. when the data acquisition process is inactive.)

When the keys (.) and (.END) are depressed, the

"S" stops flashing in the display if the data acquisition process was active. An "E" appears in the display indicating this function has executed. Once all data in the bubble memory system has been transmitted and transmission has been stopped, ("END OF FILE" appears on the terminal), "End" appears in the display.

"Error" is displayed if another function is active. "TrY TX.SA" is displayed if the transmission process is not active indicating that the user has two options; the user should use the (TX) function or he should use the (SAVE) function.

(SAVE) Save. This function is used to save the current status of the data in the bubble memory system. This status is stored in the bubble memory system status page located within the bubble memory itself. The function is used just before power down and allows any data in the bubble memory to be available again after power up. The function is executed by depressing the (SAVE) key. After the status has been saved, "SAVED" is displayed.

"Error" is displayed if another function is already active. This function can be executed only if the data acquisition process and the transmission process are not active. "USE .HS" is displayed if the user must first use the (.HS) function. "USE .noTX" is displayed if the user must first use the (.noTx) function.

Typical Operation

The following is an example of how to operate the data acquisition and transmission system under typical conditions.

keystrokes:	display:	comments:
turn power on.	"	"
turn on terminal.	"	"
	"rEAdY"	"
		Wait until terminal is warmed up and "rEAdY" is displayed. Rotary switch should be set on MICRO and thumbwheel of modem set on "19".

turn on modem.	"rEAdY"	"	Wait for MTS header.
(.),(.SIG)	"SIG"	"	Signon to MTS.
	"SiGnEd.on"	"	Wait for this display.
(CLR)	"	"	Clear display.
(0),(1)	"09"	"	Enter hours.
(3),(2)	"0932"	"	Enter minutes.
(2),(5)	"093225"	"	Enter seconds.
(.),(.TS)	"093225TS"	"	Set current time.
(.),(.CLB)	"MT. BS?"	"	Empty bubble.
(A),(1)	"MT. BS?A1"	"	
	"B.S. MT"	"	
(SEL)	"SEL."	"	Select 4 Hz. resample
(0),(4)	"SEL.04"	"	with \$COPY.
	"SEL. 10 4"	"	
(TX)			Start Transmission.
	"SEndinG"	"	

At this point any amount of descriptive information may be entered into the file R from the terminal. The first line entered must contain only a "1" in the first column. All the following lines of descriptive data,(except the last line), must start with a "1" in the first column and should contain fewer than 46 characters. The last line of descriptive data must have a zero or a blank as its first character,(in the first column).

keystrokes: display: comments:

			Subject transducer of Ch. 5 to first calibration level.
(CAL)	"CAL."	"	Enter first calibration
(C),(5)	"CAL.C5"	"	level.
	"CAL. 10 4"	"	Wait for this display.
			Subject transducer of Ch. 5 to second calibration level.
(CAL)	"CAL."	"	Enter second calibration
(D),(5)	"CAL.C5"	"	level.
	"CAL. 10 4"	"	Ch. 5 is now calibrated.
			Any other channels should be calibrated at this point.
(CLR)	"	"	Clear display.
(SS)	"	S "	Start data acquisition.
	"	"	The transmission of sampled data can be monitored from the terminal. "S" flashes in the display.

(.HS)	"	S "	Sampling may be halted to avoid the acquisition of meaningless or redundant data. This reduces the amount of data that is transmitted and accumulated in file R and may save space in the bubble memory. When transmission stops, more descriptive data may be entered at the terminal.
(SS)	"	"	Restart data acquisition.
	"	S "	
		.	
		.	
		.	
(CLR)	"	"	
(1),(0)	"10	S "	Enter a flag.
(Flag)	"1010	"	The flag will eventually appear in the last two characters of a line when it is displayed on the terminal.
		.	
		.	
		.	
(.),(.END)	"E	"	Stop the data acquisition process.
WAIT	"E	"	Wait until remainder of data in bubble is transmitted.
	"End	"	All data is now transmitted. Transmission has stopped.
(SAVE)	"SAVED	"	Recommended to log any possible bubble memory system errors.

Save data in file R.

Sign off on terminal. Type in "SIG(RETURN)".

Turn off modem.

Turn off power to the data acquisition and transmission system.

Turn off terminal.

Special Modes of Operation

As shown in Table 1, data may be acquired for some time while the transmission process is not active. This allows the data acquisition and transmission system to operate in "special" modes. After a large amount of data has been transmitted the user may be running short of disk file space on his MTS account. Provided there is still enough space in the bubble memory to store a few minutes worth of sampled data, the user may stop the transmission process while

allowing the data acquisition process to continue. Then the user can use the terminal to run MTS programs to place the data in file R on magnetic tape. (See MTS reference material.) (One page of disk file space will hold about 70 lines of sampled data or 17.5 seconds of data sampled at a rate of 4 hertz.)

After any amount of data has been transmitted to MTS, the user may stop the transmission process and analyse this data. This analysis would be done by a program run under MTS from the terminal and may be done while the data acquisition process is still active. The results of the analysis may be transmitted back to the terminal, a device in series with it, or a device connected to another line from the Amdahl computer.

As the user may use a program called PROCESS, running under MTS, to receive data transmitted by the data acquisition system, flags may be used to cause the program to perform some type of analysis.

Under certain circumstances, the user may choose, or may be forced, to acquire an amount of data first and then transmit it at a later time. (For example, MTS may not be operating.) After the data acquisition process is halted, using the (SAVE) function will allow data to be retained in the bubble memory without power. Later when MTS comes up, this data may be transmitted. Provided that the (SAVE) or the (SS) functions are not used again after or during the time this data is transmitted, the data may be transmitted again each time after the power is turned on to the data acquisition and transmission system.

Problems

Occasionally, a few problems arise in the operation of the data acquisition and transmission system.

If the automatic signon function does not work the user should try to signon to MTS manually from the terminal. Remember the address BF7A in ROM8 (EPROM #8) must be the location of a valid user I.D. coded as 4 bytes of ASCII. Also address BF82 must be the location of the appropriate pass word coded as 6 bytes of ASCII. The terminal must be set for the full duplex mode, 9600 baud, 1 stop bit, and even parity.

Once the user has signed on, if the (TX) function does not appear to work, it is because the user has not created the file, R. If this is the case, the user should abort the (TX) function by entering (.),(.noTx) at the keypad and then enter the command, "CRE R(RETURN)", at the terminal.

When data is transmitted, it is always added to the end of file R, after whatever the file R may already contain. It is the user's responsibility to take the data transmitted to file R and store it somewhere in an organized way (on magnetic tape) before the next data acquisition session begins.

During the use of the data acquisition and transmission system, A.C. power line transients may cause the generation of faulty characters, parity errors, and attention interrupts in the transmission of data to MTS. These transients may also cause a failure in the bubble memory system. The transients can be removed through the use of the appropriate power line conditioning equipment. Parity errors result in the loss of lines of sampled data. Attention interrupts also cause the transmission process to stop. Transmission may be restarted by entering (.), (noTx) to abort the current transmission process and then (TX) to restart transmission. If the bubble memory should fail, check the bubble memory boot loop, (enable the system development functions and see the "BPK 72 Bubble Memory Prototype Kit User's Manual" for instructions.) The second subroutine stored in ROM5 may be used to rewrite the boot loop.

Appendix B -- Microcomputer Software Listing

ROM1

M68SAM is the property of Motorola Spd, Inc.
Copyright 1974 by Motorola Inc.

Motorola M6800 Cross Assembler, Release 1.1

NAM ROM1 BASIC SYSTEM ROUTINES

```

*
* SOURCE FILE LABELS
*
*
* PIA LABELS
*
8400 DRA EQU $8400 -
8401 CRA EQU $8401 I
8402 DRB EQU $8402 I
8403 CRB EQU $8403 > PIA #1 & #2
8404 DRC EQU $8404 I
8405 CRC EQU $8405 I
8406 DRD EQU $8406 I
8407 CRD EQU $8407 -
*
*
* LATCH LABELS. ( SN74LS273 )
*
8800 TRH EQU $8800 - ADDRESS TRAP LABELS
8801 TRL EQU $8801
8C00 LCON EQU $8C00 - CONTROL LATCH
*
*
* PROGRAMMABLE KEYBOARD DISPLAY INTERFACE (I8279-5)
*
8001 PKDICR EQU $8001 - CONTROL REGISTER; WRITE ONLY
8001 PKDISR EQU $8001 - STATUS REGISTER; READ ONLY
8000 FIFO EQU $8000 - KEYBOARD INPUT RAM; READ ONLY
8000 DISRAM EQU $8000 - DISPLAY RAM; READ/WRITE
*
*
* MONITOR GLOBAL VARIABLES AND FLAGS
*
0000 READEN EQU $0000 - READ ENABLE
0001 STFLG EQU $0001 - STACK FLAG
0002 ENTRYC EQU $0002 - ENTRY COUNTER
0004 ENTRY1 EQU $0004 - ENTRIES VIA ENTR FUNCTION
0005 ENTRY2 EQU $0005
0006 ENTRY3 EQU $0006
0007 ENTRY4 EQU $0007
0008 ENTRY5 EQU $0008
0009 ENTRY6 EQU $0009
000A ENTRY7 EQU $000A
000B ENTRY8 EQU $000B

```



```

000C  LISTC  EQU $000C
000D  IXRV   EQU $000D,E -X REG. STORAGE, M. L. ONLY
000F  GOTOFE EQU $000F - GO TO FUNCTION ENABLE FLAG
0010  MONFLG EQU $0010 - IND. M.L. STOPPED BY MONITOR
0011  SECFLG EQU $0011 - IND. 2ND KEYBRD. FTM. SELECTED
0012  LEDNO  EQU $0012 - # OF LED TO BE WRITTEN NEXT
      *      ( FIRST LED# = 0 )
      *
      *

```

```

*  LINKAGE TABLE ENTRIES FOR SWI  ROUTINES
*

```

```

F400  IERR   EQU $F400
F402  ISTX1  EQU $F402
F404  ISTX2  EQU $F404
F406  ISABX  EQU $F406
F408  ILABX  EQU $F408
      *
      *

```

```

*  JUMP TABLE ENTRIES FOR  ROUTINES
*

```

```

FBF0  TRAP   EQU $FBF0
FBF3  STOP   EQU $FBF3
F7E9  RM3RST EQU $F7E9
FBF9  PKDIRS EQU $FBF9
      *

```

```

*****

```

```

*  MONITOR PROGRAMS
*

```

```

FC00                                ORG      $FC00
      *

```

```

*  GENERAL SYSTEM RESET
*

```

```

FC00 01          RESET  NOP
FC01 8E 03FF          LDS #$03FF - SET STACK PTR.
FC04 BD FDF6          JSR CNPIA1 - CONFIGUR PIA'S 1 & 2
FC07 FF              FCB $FF,$FF,$FF,$FF - ALL OUTPUTS
FC08 FF
FC09 FF
FC0A FF
FC0B BD FBF9          JSR PKDIRS - CONFIGURE PKDI
FC0E CE 0020          LDX #$0020
FC11 6F 00  RSL1      CLR 0,X      - CLEAR RAM $00 TO $20
FC13 09              DEX
FC14 26 FB          BNE RSL1
FC16 7C 0000          INC READEN - ENABLE KEYBOARD
FC19 7C 000F          INC GOTOFE -DISABLE NO-BEBUG "GOTO"
FC1C 7E F7E9          JMP RM3RST - CONTINUE WITH RESET
FC1F CE FCB2  RM1RST  LDX #DSPTST
FC22 DF 0D          STX      IXRV
FC24 96 0D          LDA A      IXRV
FC26 D6 0E          LDA B      IXRV+1
FC28 BD FEFB          JSR      WRITE4

```



```

FC2B 7C 0010      INC MONFLG - IND.MON.ACTIVE,M.L.NOT
FC2E 0E           CLI          -  ENABLE INTERRUPTS
FC2F 3E           RESWAI WAI
FC30 20 FD        BRA          RESWAI
FC32 01           NOP

```

```

*   END OF RESET *

```

```

*-----*-----*

```

```

*   PROGRAMS TO TEST HARDWARE

```

```

*

```

```

*   PIA TEST PROGRAM

```

```

*

```

```

*   - ENTERED AFTER RESET IF A BRANCH IS NOT INSERTED
*   - GENERATES A 1 HZ SQUARE WAVE ON PIA 1 & 2 OUTPUTS
*   - AN INFINITE LOOP

```

```

*

```

```

FC33 53           PIATST COM B
FC34 F7 8400      STA B      DRA - STORE B IN PIA OUTPUTS
FC37 F7 8402      STA B      DRB
FC3A F7 8404      STA B      DRC
FC3D F7 8406      STA B      DRD
FC40 BD FE2E      JSR        DELAY2
FC43 00F9         FDB        $00F9 - DELAY .5 SEC
FC45 20 EC        BRA        PIATST

```

```

*   END   *

```

```

*

```

```

*   NMI TEST PROGRAM

```

```

*   - 2 HZ SQUARWAVE AT PIA OUTPUTS
*   - 10 CYCLES, ( 5 SEC. )
*   - INTERRUPT MASK CLEARED

```

```

*

```

```

FC47 01           NMITST NOP
FC48 C6 14        LDA B      #20          - 20 ITERATIONS
FC4A 43           NMIL1 COM A
FC4B 8D 24        BSR        OUTPUT
FC4D 36           PSH A          **
FC4E BD FE2E      JSR        DELAY2
FC51 007C         FDB        $007C      - DELAY .25 SEC.
FC53 32           PUL A          **  A AS BEFORE
FC54 5A           DEC B
FC55 26 F3        BNE        NMIL1
FC57 32           PUL A          ** - A IS MAIN CCR
FC58 84 EF        AND A      #%11101111 - CLEAR INT. MASK
FC5A 36           PSH A          ** - MAIN CCR O.K.
FC5B 01           NOP
FC5C 3B           RTI

```

```

*   END   *

```

```

*

```

```

*   IRQ TEST PROGRAM

```

```

*   - 4 HZ SQUARE WAVE AT PIA OUTPUTS
*   - 20 CYCLES, ( 5 SEC. )

```

```

*

```

```

FC5D 01           IRQTST NOP
FC5E C6 28        LDA B      #40          - 40 ITERATIONS
FC60 86 F0        LDA A      #$F0

```



```

FC62 43          IRQL1 COM A
FC63 8D 0C        BSR      OUTPUT
FC65 36          PSH A      **
FC66 BD FE2E      JSR      DELAY2
FC69 003E        FDB      $003E - .125 SEC. DELAY
FC6B 32          PUL A      **
FC6C 5A          DEC B
FC6D 26 F3        BNE      IRQL1
FC6F 01          NOP
FC70 3B          RTI
*   END   *
*
*   PIA OUTPUT SUBROUTINE
*   - ALL PORTS ON PIA 1 & 2 ASSUMED TO BE OUTPUTS.
*
FC71 B7 8400 OUTPUT STA A   DRA - DATA IN ACC.A IS O/P
FC74 B7 8402      STA A   DRB
FC77 B7 8404      STA A   DRC
FC7A B7 8406      STA A   DRD
FC7D 39          RTS
*   END   *
*
*   PROGRAM TO TEST THE LATCHES AND THE ADDRESS TRAP
*   - LATCHES SET TO "1" FOR .5 SEC. AND "0" FOR
*     .25 SEC. FOR 20 CYCLES
*   - ADDRESS TRAP PRODUCES A NMI
*
FC7E C6 14      LATTST LDA B   #$14 - 20 ITERATIONS
FC80 01          NOP          -RESERVED FOR "CLI"
FC81 CE FFFF LATTL1 LDX      #$FFFF
FC84 86 FF      LDA A   #$FF
FC86 FF 8800     STX TRH   - SET TRAP ADDRESS
FC89 B7 8C00     STA A   LCON - SET CONTROL LATCH
FC8C BD FE2E     JSR      DELAY2
FC8F 00F9        FDB      $00F9 - DELAY FOR .5 SEC.
FC91 7F 8800     CLR      TRH   - CLEAR LATCHES
FC94 7F 8801     CLR      TRL
FC97 7F 8C00     CLR      LCON
FC9A BD FE2E     JSR      DELAY2
FC9D 007C        FDB      $007C - DELAY .25 SEC.
FC9F 5A          DEC B
FCA0 26 DF        BNE      LATTL1
FCA2 CE FCAE     LDX      #LATA1
FCA5 FF 8800     STX TRH   -SET TRAP FOR ADD. "LATA1"
FCA8 86 01        LDA A   #$01
FCAA B7 8C00     STA A   LCON - ENABLE TRAP(EN. NMI)
FCAD 5A          DEC B
FCAE 5A          LATA1 DEC B   - WHERE NMI OCCURS.
FCAF 5A          DEC B
FCB0 20 CC        BRA      LATTST
*   END   *
*
*   PROGRAM TO TEST THE DISPLAY
*   - CHECKS THE "ERROR" ROUTINE VIA A SWI

```



```

* - CHECKS THE DISPLAY BY DISPLAYING A TEST TABLE
* - CHECKS THE DISPLAY SUBROUTINES, "BITODS", "READ4",
*   DSTOBI
*
FCB2 3F          DSPTST SWI          - DISPLAY "ERROR"
FCB3 0000          FDB 0,IERR
FCB5 F400
FCB7 CE FF60      LDX #DATTBL -(X)=ADD. OF TEST TABLE
FCBA C6 10        LDA B  #$10      - 16 ITERATIONS
FCBC 01           NOP
FCBD 01           NOP
FCBE 01           NOP
FCBF 37          DSPTL1 PSH B          **
FCC0 BD FE42      JSR CLDISP - CHECK DISPLAY, TABLE
FCC3 A6 01        LDA A  1,X - GET DISPLAY PATTERN
FCC5 16           TAB              - COPY A TO B
FCC6 08           INX
FCC7 08           INX
FCC8 08           INX
FCC9 DF 0D        STX IXRV          ***
FCCB BD FF2E      JSR DTSR1
FCCE BD FE2E      JSR DELAY2
FCD1 01FF        FDB $01FF        - 2 SEC DELAY
FCD3 DE 0D        LDX IXRV          ***
FCD5 33           PUL B            **
FCD6 5A           DEC B
FCD7 26 E6        BNE DSPTL1
FCD9 BD FE42      JSR CLDISP - GET READY FOR NEXT
FCDC BD FE2E      JSR DELAY2
FCDF 02FF        FDB $02FF - DELAY OF 1.5 SEC.
*
FCE1 4F          CLR A      - TEST DISPLAY ROUTINES
FCE2 36          DSPTL2 PSH A          **
FCE3 BD FEEA      JSR BITODS - A TO DISP. IN A,B
FCE6 BD FF2E      JSR DTSR1 - FILL DISP. WITH A,B
FCE9 BD FE2E      JSR DELAY2
FCEC 01FF        FDB $01FF        1 SEC. DELAY
FCEE 4F          CLR A
FCEF BD FE91      JSR READ4 4 BYTES FROM DISP.
*              RESULT IN A,B AS BINARY
FCF2 30          TSX
FCF3 A1 00        CMP A  0,X - VALUE UNCHANGED
FCF5 27 0B        BEQ DSPTS1 - SKIP IF "YES"
FCF7 CE FD0B      LDX #READ4E
FCFA BD FF47      JSR DTSR2 - DISPLAY "ERROR"
FCFD BD FE2E      JSR DELAY2
FD00 03FF        FDB $03FF        - DELAY 2 SEC
FD02 BD FE42      DSPTS1 JSR CLDISP - READY TO CONTINUE
FD05 32           PUL A            **
FD06 4A           DEC A
FD07 26 D9        BNE DSPTL2
FD09 20 A7        BRA DSPTST
*   END   *
FD0B 12          READ4E FCB  $12      %00010010 R

```


FD0C	7A	FCB	\$7A	%01111010	E
FD0D	77	FCB	\$77	%01110111	A
FD0E	1F	FCB	\$1F	%00011111	D
FD0F	27	FCB	\$27	%00100111	4
FD10	00	FCB	\$00	BLANK	
FD11	7A	FCB	\$7A	E	
FD12	92	FCB	\$92	%10010010	R.

*

*

* PROGRAM TO TEST RAM

* - WHEN AN ERROR OCCURS, THE FAULTY ADDRESS IS SENT
 * TO THE DISPLAY RAM ALONG WITH THE DATA STORED AND
 * THE DATA RETREIVED.

*

FD13	CE	FD81	RTST	LDX	#RAMTEST	
FD16	BD	FF47		JSR	DTSR2 - DISPLAY "RAMTEST"	
FD19	86	FF		LDA A	#\$FF - DATA TO BE STORED	
FD1B	CE	17FF		LDX	#\$17FF - STARTING LOCATION	
*						
FD1E	6F	00	RTSTL1	CLR	0,X - CLEAR RAM LOCATION	
FD20	A7	00		STA A	0,X - CHANGE 0'S TO 1'S	
FD22	E6	00		LDA B	0,X - RECALL	
FD24	C1	FF		CMP B	#\$FF - RECALLED VALUE O.K.?	
FD26	27	07		BEQ	RTSTS1 - SKIP IF YES	
FD28	BD	FDE2		JSR	DSPXAB- DISP. LOC., X & A,B	
FD2B	01			NOP		
FD2C	3E			WAI	- STOP	
FD2D	20	E4		BRA	RTST	
FD2F	6F	00	RTSTS1	CLR	0,X - CHANGE 1'S TO 0'S	
FD31	6D	00		TST	0,X - ALL 0'S?	
FD33	27	0A		BEQ	RTSTS2 - SKIP IF YES	
FD35	4F			CLR A	- A = VALUE STORED	
FD36	E6	00		LDA B	0,X - B = FAULTY VALUE	
FD38	BD	FDE2		JSR	DSPXAB - DISP. LOC., X & A,B	
FD3B	01			NOP		
FD3C	3E			WAI	- STOP	
FD3D	20	D4		BRA	RTST	
FD3F	09		RTSTS2	DEX	- DECREMENT RAM LOC.	
FD40	26	DC		BNE	RTSTL1	
*						
FD42	CE	17FE	RTST2	LDX	#\$17FE - LOC. OF 2ND TEST	
FD45	EF	00	RTSTL2	STX	0,X - STORE EVERY SECOND	
FD47	09			DEX	LOC. IN THAT LOC.	
FD48	09			DEX	- DECREMENT LOC. BY 2	
FD49	26	FA		BNE	RTSTL2	
FD4B	CE	17FE		LDX	#\$17FE - LOCATION OF TEST	
FD4E	AC	00	RTSTL3	CPX	0,X - CHECK	
FD50	27	1D		BEQ	RTSTS3 - SKIP IF O.K	
FD52	A6	00		LDA A	0,X - GET 1ST BYTE OF ERR.	
FD54	E6	01		LDA B	1,X - GET 2ND BYTE OF ERR.	
FD56	BD	FDE2		JSR	DSPXAB - DISP. LOC., X & ERR.	
FD59	86	63		LDA A	#\$63 - READ DISP. RAM 3	
FD5B	B7	8001		STA A	PKDICR WITHOUT AUTOINC.	
FD5E	F6	8000		LDA B	DISRAM - DISPLAY CODE IN B	


```

FD61 CA 80          ORA B   #$80   - ADD DECIMAL POINT
FD63 86 83          LDA A   #$83   - DISP. NEW CODE
FD65 B7 8001        STA A   PKDICR
FD68 F7 8000        STA B   DISRAM
FD6B 01            NOP
FD6C 3E            WAI
FD6D 20 D3          BRA     RTST2   - STOP
FD6F 09            DEX          - DEC. LOC. OF TEST
FD70 09            DEX
FD71 26 DB          BNE     RTSTL3
*
FD73 CE FD89        LDX     #RAM104 - DISPLAY "RAM 10-4"
FD76 BD FF47        JSR     DTSR2
FD79 BD FE2E        JSR     DELAY2
FD7C 07FF          FDB     $07FF   - DELAY 4 SEC
FD7E 7E FD13        JMP     RTST
*   END   *

```

```

*
FD81 12            RAMTES FCB     $12      %00010010   R
FD82 77            FCB     $77      %01110111   A
FD83 16            FCB     $16      %00010110   N
FD84 86            FCB     $86      %10000110
FD85 45            FCB     $45      %01000101   T
FD86 7A            FCB     $7A      %01111010   E
FD87 6E            FCB     $6E      %01101110   S
FD88 45            FCB     $45      %01000101   T
*

```

```

FD89 12            RAM104 FCB     $12      %00010010   R
FD8A 77            FCB     $77      %01110111   A
FD8B 16            FCB     $16      %00010110   N
FD8C 86            FCB     $86      %10000110
FD8D 00            FCB     $00      BLANK
FD8E 05            FCB     $05      %00000101   1
FD8F FD            FCB     $FD      %11111101   0.
FD90 27            FCB     $27      %00100111   4
*

```

*

SUBROUTINES

*

*

```

FD91    DSPSN  EQU    *          DISPLAY SIGN

```

*

```

* - DISPLAYS A "-" IF N IN CCR IS SET
* - ALSO DISPLAYS A "O" IN SAME LED IF V IS SET
* - CHANGES A,B

```

*

```

FD91 07            TPA          - STORE CCR
FD92 2B 03          BMI     DSPSN1
FD94 5F            CLR B       - NO NEGATIVE
FD95 20 02          BRA     DSPSN2
FD97 C6 02          DSPSN1 LDA B   #$02
FD99 06            DSPSN2 TAP          - RESTORE CCR
FD9A 28 02          BVC     DSPSN3

```



```

FD9C CA 1E          ORA B   #$1E   - OVERFLOW
FD9E F7 8000 DSPSN3 STA B   DISRAM - WRITE SIGN
FDA1 06             TAP           - RESTORE CCR
FDA2 39             RTS
**  END    **
*
*
      FDA3      DSPDP   EQU *   DISPLAY DECIMAL POINT
*
* - DISPLAYS A DECIMAL POINT AFTER THE CHARACTER
*   SPECIFIED BY ACC. B
* - CHANGES A,B,CCR
*
FDA3 C4 07          AND B   #$07   - DISRAM LOC. READ
FDA5 86 60          LDA A   #$60   - NO AUTOINCREMENT
FDA7 1B             ABA
FDA8 B7 8001        STA A   PKDICR
FDAB B6 8000        LDA A   DISRAM - GET ORIGINAl CHAR.
FDAE 8A 80          ORA A   #$80   - ADD DECIMAL POINT
FDB0 CB 90          ADD B   #%10010000 - AUTOINC. WRITE
FDB2 F7 8001        STA B   PKDICR - REPLACE CHAR.
FDB5 B7 8000        STA A   DISRAM
FDB8 39             RTS
**  END    **
*
*
      FDB9      FLSH    EQU *           FLASH
*
* - CAN BE USED TO FLASH AN LED WITH AN
*   ARBITRARY CHARACTER
* - CALLING SEQUENCE:
*   JMP FLSH
*   FCB (DISPLAY PATTERN),(LED #, 0 TO 7 )
* - CHANGES A,B,CCR,X
*
FDB9 30             TSX           X PTS. TO RETURN VECTOR
FDBA EE 00          LDX         0,X  X IS RETURN LOCATION
FDBC 86 60          LDA A   #$60
FDBE AB 01          ADD A   1,X   ADD LED# TO READ INST.
FDC0 B7 8001        STA A   PKDICR
FDC3 F6 8000        LDA B   DISRAM READ PATTERN
FDC6 26 04          BNE      FLSHS1 BRANCH IF NOT CLEAR
*
FDC8 E6 00          LDA B   0,X   IF LED WAS CLEAR, SET IT
FDCA 20 01          BRA      FLSHS2
*
FDCC 5F             FLSHS1 CLR B   IF LED WAS SET, CLEAR IT
FDCD 88 F0          FLSHS2 EOR A   #$F0 CHANGE READ TO WRITE
FDCF B7 8001        STA A   PKDICR
FDD2 F7 8000        STA B   DISRAM WRITE PATTERN
FDD5 96 12          LDA A   LEDNO GET # OF LED TO BE
FDD7 84 07          AND A   #$07 WRITTEN INTO NEXT
FDD9 8B 90          ADD A   #$90
Fddb B7 8001        STA A   PKDICR READY TO WRITE

```



```

FDDE 31          INS
FDDF 31          INS      ADJUST STACK
FDE0 6E 02      JMP      2,X      RETURN
*      79USEC.
**      END      **
*
*
*      DSPXAB      - DISPLAY X, A, AND B
*      - DISPLAYS THE CONTENTS OF A,B, AND X
*      - CHANGES A,B, AND X
*
FDE2 36          DSPXAB PSH A      ** CONTENTS OF A
FDE3 37          PSH B      *** CONTENTS OF B
FDE4 BD FE42     JSR      CLDISP  - PREP. FOR WRITE
FDE7 DF 0D       STX      IXRV
FDE9 96 0D       LDA A      IXRV  - A = XH
FDEB D6 0E       LDA B      IXRV+1 - B = XL
FDED BD FEFB     JSR      WRITE4
FDF0 33          PUL B      *** RESTORE B
FDF1 32          PUL A      ** RESTORE A
FDF2 BD FEFB     JSR      WRITE4  - DISPLAY ERROR
FDF5 39          RTS
*      END      *
*
*
*      CONFIGURE PIA'S 1 & 2
* - OUTPUTS OF PIA'S 1 & 2 CONFIGURED AS DESIGNATED
* BY THE 4 BYTE FCB AFTER "JSR CNPIA1".
*      - NO PIA IRQ CAPABILITY
*      - CALLED AS FOLLOWS:      JSR CNPIA1
*                                  FCB $XX,$XX,$XX,$XX
*                                  NEXT INSTRUCTION
*
*      - CHANGES X,A
*
FDF6 7F 8401 CNPIA1 CLR      CRA  - ACCESS TO DDR'S
FDF9 7F 8403      CLR      CRB
FDFC 7F 8405      CLR      CRC
FDFF 7F 8407      CLR      CRD
FE02 30          TSX      (X)= RETURN ADD.LOC.
FE03 EE 00       LDX      0,X  (X)= RETURN ADD.(FCB)
FE05 A6 00       LDA A      0,X - FIRST DDR VALUE
FE07 B7 8400     STA A      DRA
FE0A A6 01       LDA A      1,X
FE0C B7 8402     STA A      DRB
FE0F A6 02       LDA A      2,X
FE11 B7 8404     STA A      DRC
FE14 A6 03       LDA A      3,X
FE16 B7 8406     STA A      DRD
FE19 86 04       LDA A      #$04 - IRQ'S DIS.,DATA REG.
FE1B B7 8401     STA A      CRA  ACCESS
FE1E B7 8403     STA A      CRB
FE21 B7 8405     STA A      CRC
FE24 B7 8407     STA A      CRD

```



```

FE27 B7 8407      STA A   CRD
FE2A 32           PUL A           - DECREMENT SP BY TWO
FE2B 32           PUL A
FE2C 6E 04        JMP      4,X   - RETURN TO 4+RET.ADD.
*   END   *
*
*
*   A TIME DELAY PROGRAM
* - CALLED VIA      JSR DELAY2
*                   FDB $"DELAY FACTOR"
*                   NEXT INSTRUCTION
* - TIME DELAY = (38 + 2058(DELAY FACTOR))TCYCLE
* - EG) .5 SEC= 5 EE 5      SO  DF=2.5 EE 2=$00F9
*   - CHANGES A,X,CCR
*
FE2E 30           DELAY2 TSX           RETURN ADDRESS LOCATION
FE2F EE 00         LDX      0,X RETURN ADDRESS,(FDB LOC)
FE31 EE 00         LDX      0,X VALUE AT RETURN ADDRESS
FE33 4F           DLL1  CLR A   - LOOP1, FDB ITERATIONS
FE34 4A           DLL2  DEC A   - NESTED LOOP2, 256 ITER.
FE35 01           NOP      - DELAY
FE36 26 FC        BNE      DLL2
FE38 09           DEX
FE39 26 F8        BNE      DLL1
FE3B 30           TSX           RETURN ADDRESS LOCATION
FE3C EE 00         LDX      0,X RETURN ADDRESS
FE3E 32           PUL A   - INC. STACK BY TWO
FE3F 32           PUL A
FE40 6E 02        JMP      2,X - RETURN
*   END   *
*
*
*   PROGRAM TO CLEAR THE DISPLAY RAM
* - BLANKS THE DISPLAY
* - CHANGES REGISTERS A,B,CCR
* - CLEARS FIFO STATUS IN PKDI
* - CLEARS SECFLG AS THE SECOND FUNCTION
*   LED IS ALSO CLEARED
*
FE42 86 00        CLDISP LDA A   #$00 8 8BIT CHAR. DISP.,L.ENT.
FE44 B7 8001      STA A   PKDICR ENCODED SCAN KEYBD.
FE47 86 3F        LDA A   #$3F PROGRAM CLOCK; 32.3KHZ
FE49 B7 8001      STA A   PKDICR
FE4C C6 08        LDA B   #$08 - 8 ITERATIONS
FE4E 86 80        LDA A   #$80 - WRITE TO BYTE 0
FE50 B7 8001      CLDPL1 STA A   PKDICR WITHOUT AUTOINCREMENT
FE53 7F 8000      CLR      DISRAM
FE56 4C           INC A
FE57 5A           DEC B
FE58 26 F6        BNE      CLDPL1
FE5A 86 90        LDA A   #$90 - WRITE TO BYTE 0
FE5C B7 8001      STA A   PKDICR WITH AUTOINCREMENT
FE5F 7F 0011      CLR      SECFLG
FE62 7F 0012      CLR      LEDNO - NEXT WRITE TO LED#0

```



```

FE65 39          RTS
*   192 USEC.
*   END   *
*
*   DISPLAY TO BINARY SUBROUTINE
* - USES "DATTBL" TO CONVERT A DISPLAY PATTERN BYTE IN
*   A TO A BINARY NIBBLE,( LOWER NIBBLE ), IN ACC. A.
* - IF THE HEX DIGIT DISPLAY PATTERN IS NOT FOUND
*   THE CARRY FLAG IS SET. OTHERWISE IT IS CLEARED.
* - B,X ARE CHANGED, A MAY BE CHANGED.
*
FE66 CE FF60 DSBISR LDX      #DATTBL LOC. OF DATTA TABLE
FE69 C6 10          LDA B    #$10      - 16 HEX DIGITS
FE6B A1 01          DSBIL1 CMP A    1,X
FE6D 27 08          BEQ      AFND
FE6F 08            INX
FE70 08            INX
FE71 08            INX      - INC. TO NEXT PATTERN
FE72 5A            DEC B
FE73 26 F6          BNE      DSBIL1
FE75 0D            SEC      -PATTERN NOT FOUND,
FE76 39            RTS      CARRY SET
FE77 0C            CLC      - DISPLAY PATTERN FOUND
FE78 A6 02          LDA A    2,X      - LOAD BINARY NIBBLE,
FE7A 39            RTS      CARRY CLEAR
*   END   *
*
*   DISPLAY PATTERN TO BINARY CONVESION PROGRAM
* - CONVERTS TWO BYTES OF DISPLAY CODE IN A,B
*   INTO ONE BYTE OF BINARY CODE STORED IN ACC. A .
* - A,B  = MOST,LEAST SIGNIFICANT CHARACTER PATTERN
* - AN IMPROPER DISPLAY PATTERN SETS THE CARRY.
*   OTHERWISE THE CARRY IS CLEARED.
*
FE7B 37            DSTOBI PSH B      ** 2ND CHAR.
FE7C 8D E8          BSR      DSBISR  - A TO BINARY NIB.
FE7E 24 02          BCC      DSBIS1  - SKIP IF A FOUND
FE80 33            PUL B      - A FAULTY
FE81 39            RTS
FE82 48            DSBIS1 ASL A
FE83 48            ASL A      MAKE BINARY NIBBLE
FE84 48            ASL A      HIGH NIB.
FE85 48            ASL A
FE86 33            PUL B      ** - GET 2ND CHAR.
FE87 36            PSH A      *** UPPER NIBBLE
FE88 17            TBA
FE89 8D DB          BSR      DSBISR  - 2ND CHAR. TO NIB.
FE8B 33            PUL B      *** UPPER NIBBLE
FE8C 24 01          BCC      DSBIS2  - 2ND CHAR. FOUND?
FE8E 39            RTS      - NOT FOUND
FE8F 1B            DSBIS2 ABA      ADD UPPER TO LOWER
FE90 39            RTS      CARRY CLEAR

```



```

*   END   *
*
*
*   SUBROUTINE TO READ FOUR HEX. CHARACTERS
* - READS THE FIRST 4 DISPLAY PATTERN BYTES IN
*   THE DISPLAY RAM STARTING FROM THE LED NO. IN A
*   AND CONVERTS THEM TO THEIR TWO BYTE BINARY
*   VALUE STORED IN ACC A,B.
* - USES SUBROUTINES DSTOBI AND DSBISR
* - CARRY IS CLEAR IF DISPLAY PATTERNS READ WERE FOUND
* - CHANGES X,A,B
* - (MSB,LSB) = (A,B)
*
FE91 8B 70    READ4  ADD A    #%01110000
FE93 B7 8001    STA A    PKDICR - PREP. TO READ
FE96 B6 8000    LDA A    DISRAM - READ FIRST CHAR.
FE99 F6 8000    LDA B    DISRAM - READ 2ND CHAR.
FE9C BD FE7B    JSR      DSTOBI - CONVERT CHARS.
FE9F 25 10      BCS      RD4RTS - RETURN IF NO GOOD
FEA1 36         PSH A      ** - STORE MSB
FEA2 B6 8000    LDA A    DISRAM - READ 3RD CHARACTER
FEA5 F6 8000    LDA B    DISRAM - READ 4TH CHARACTER
FEA8 BD FE7B    JSR      DSTOBI - CONVERT CHARS.
FEAB 24 02      BCC      RD4S1 - SKIP IF NO GOOD
FEAD 32         PUL A      ** RESTORE STACK
FEAE 39         RTS        - RETURN
FEAF 16         RD4S1 TAB   - TRANSFER LSB TO B
FEB0 32         PUL A      ** MOST SIGNIFICANT BYTE
FEB1 39         RD4RTS RTS
**   END   **
*
*   SUBROUTINE TO READ TWO HEX. CHARACTERS
* - READS THE FIRST 2 DISPLAY PATTERN BYTES IN THE
*   DISPLAY RAM STARTING FROM THE LED. NO. IN A AND
*   CONVERTS THEM TO THEIR
*   ONE BYTE BINARY VALUE STORED IN ACC A
* - USES SUBROUTINES DSTOBI AND DSBISR
* - CARRY IS CLEAR IF DISPLAY PATTERNS READ WERE FOUND
* - CHANGES X,A,B
* - (MSB,LSB) = (A,B)
*
FEB2 8B 70    READ2  ADD A    #%01110000
FEB4 B7 8001    STA A    PKDICR - PREPARE TO READ
FEB7 B6 8000    LDA A    DISRAM - READ FIRST CHAR.
FEBA F6 8000    LDA B    DISRAM - READ 2ND CHAR.
FEBD BD FE7B    JSR      DSTOBI - CONVERT CHARS
FEC0 39         RTS
*
**   END   **
*
*
FEC1      RDWL    EQU *    READ WITH LIMIT
*
* - READS A BYTE FORM THE DISPLAY RAM

```



```

* - CONVERTS IT TO A HEX NIBBLE IN LOWER NIBBLE OF A
* - IF DISP. PATTERN NOT FOUND OR HEX VALUE TOO LARGE,
*   WE INS, INS, RTI
* - ASSOCIATED WITH KEYBORD FUNCTIONS
* - CALLING SEQUENCE:
*       JSR RDWL
*       FCB (UPPER LIMIT OF HEX CHARACTER)
*
FEC1 B6 8000      LDA A   DISRAM   GET CHARACTER
FEC4 BD FE66      JSR     DSBISR   CHANGE TO HEX
FEC7 30           TSX           X PTS. TO RET.VEC.
FEC8 EE 00        LDX      0,X     X IS RETURN VECTOR,
FECA 31           INS           LOC. OF UPPER LIM.
FECB 31           INS           ADJUST STACK
FECC 25 06        BCS      RDWLX1  EXIT IF NOT FOUND
*
FECE A1 00        CMP A   0,X
FED0 2E 02        BGT      RDWLX1  EXIT IF TOO LARGE
*
FED2 6E 01        JMP      1,X     RETURN
*
FED4 BD FE42 RDWLX1 JSR      CLDISP
FED7 3B           RTI           GO BACK TO MAIN
**  END  **
*
*
*   SUBROUTINE FOR THE BINARY TO DISPLAY PROGRAM
* - USES DATTBL TO FIND THE HEX. DISPLAY PATTERN FOR
*   A BINARY NIBBLE STORED IN BITS 3 TO 0 OF ACC. A
* - CHANGES X,A,B
* - DISPALY PATTERN STORED IN ACC. B
*
FED8 CE FF60 BIDSSR LDX      #DATTBL  LOC. OF DATA TABLE
FEDB C6 10         LDA B   #$10    - 16 POSSIBLE VALUES
FEDD A1 02         BIDSL1 CMP A   2,X  - COMPARE A WITH TABLE
FEDF 27 06         BEQ      BFND    - BRANCH WHEN FOUND
FEE1 08           INX
FEE2 08           INX              - GO TO NEXT VALUE
FEE3 08           INX
FEE4 5A           DEC B
FEE5 26 F6         BNE      BIDSL1
FEE7 E6 01         BFND    LDA B   1,X  - PATTERN IN B
FEE9 39           RTS
*  END  *
*
*
*   BINARY TO DISPLAY PATTERN CONVERSION PROGRAM
* - CONVERTS A BINARY BYTE STORED IN ACC. A INTO
*   TWO BYTES OF HEX. DISP. PATTERN STORED IN A,B
* - THE MOST SIGNIFICANT CHARACTER IS STORED IN A
* - USES SUBROUTINE BIDSSR, AND TABLE DATTBL
* - CHANGES X,A,B
*

```



```

FEEA 36          BITODS PSH A          ** STORE THE WHOLE BYTE
FEEB 84 F0          AND A  #$F0 - OBTAIN HIGHER NIBBLE
FEED 44          LSR A
FEEE 44          LSR A
FEFF 44          LSR A
FEF0 44          LSR A          - SHIFT HIGHER NIB. DOWN
FEF1 8D E5        BSR      BIDSSR      - MS CHAR. IN B
FEF3 32          PUL A          ** WHOLE BINARY BYTE
FEF4 37          PSH B          ** MOST SIG. CHAR.
FEF5 84 0F        AND A  #$0F      - OBTAIN LOWER NIB.
FEF7 8D DF        BSR      BIDSS      - LEAST SIG. CHAR. IN B
FEF9 32          PUL A          ** MOST SIG. CHAR. IN A
FEFA 39          RTS

```

```

*   END   *

```

```

*

```

```

*

```

```

*   PROGRAM TO WRITE FOUR HEX. CHAR. INTO THE DISPLAY

```

```

*

```

```

* - REQUIRES A TWO BYTE BINARY VALUE STORED IN A,B

```

```

* - (MSB,LSB) = (A,B)

```

```

* - USES BITODS

```

```

* - CHANGES X,A,B

```

```

*

```

```

FEFB 37          WRITE4 PSH B          ** - STORE LSB
FEFC BD FEEA      JSR      BITODS - (A) TO PATTERN
FEFF B7 8000      STA A  DISRAM - DISPLAY 1ST CHAR.
FF02 F7 8000      STA B  DISRAM - DISPLAY 2ND CHAR.
FF05 32          PUL A          ** - RECALL LSB
FF06 BD FEEA      JSR      BITODS - (A) TO PATTERN
FF09 B7 8000      STA A  DISRAM - DISPLAY 3RD CHAR.
FF0C F7 8000      STA B  DISRAM - DISPLAY 4TH CHAR.
FF0F 96 12        LDA A  LEDNO - UPDATE LEDNO
FF11 8B 04        ADD A  #$04
FF13 97 12        STA A  LEDNO
FF15 39          RTS

```

```

**   END   **

```

```

*

```

```

*

```

```

*   PROGRAM TO WRITE TWO HEX CHAR. INTO THE DISPLAY

```

```

*

```

```

* - REQUIRES A ONE BYTE BINARY VALUE STORED IN A

```

```

* - USES BITODS

```

```

* - CHANGES X,A,B

```

```

*

```

```

FF16 BD FEEA WRITE2 JSR      BITODS - (A) TO PATTERN
FF19 B7 8000      STA A  DISRAM - DISPLAY 1ST CHAR.
FF1C F7 8000      STA B  DISRAM - DISPLAY 2ND CHAR.
FF1F 96 12        LDA A  LEDNO - UPDATE LEDNO
FF21 8B 02        ADD A  #$02
FF23 97 12        STA A  LEDNO
FF25 39          RTS

```

```

**   END   **

```

```

*

```

```

*

```



```

FF26      WRITE0 EQU *      WRITE ZERO
*
* - PREPARES THE PKDI FOR A WRITE WITH AUTOINCREMENT
* TO THE LED SPECIFIED BY A
* - CHANGES A, CCR
*
FF26 84 0F      AND A  #$0F      FORM A WRITE INSTRUCTION
FF28 8B 90      ADD A  #$90      WITH AUTOINC.
FF2A B7 8001     STA A  PKDICR    SEND TO PKDI
FF2D 39         RTS
*
**  END  **
*
*
*      DISPLAY TEST SUBROUTINE 1
* - STORES ACC. A,B ALTERNATLY IN THE DISPLAY RAM
*
FF2E B7 8000 DTSR1  STA A  DISRAM
FF31 F7 8000      STA B  DISRAM
FF34 B7 8000      STA A  DISRAM
FF37 F7 8000      STA B  DISRAM
FF3A B7 8000      STA A  DISRAM
FF3D F7 8000      STA B  DISRAM
FF40 B7 8000      STA A  DISRAM
FF43 F7 8000      STA B  DISRAM
FF46 39          RTS
*  END  *
*
*
*      DISPLAY TEST SUBROUTINE 2
* - STORES AN EIGHT BYTE PATTERN TABLE IN THE DISPLAY
* - THE LOCATION OF THE FIRST BYTE IN THE TABLE MUST
* BE STORED IN X BEFORE CALLING THIS SUBROUTINE
* - CHANGES A,B,X
*
FF47 BD FE42 DTSR2 JSR      CLDISP
FF4A C6 08      LDA B  #$08 - 8 BYTES DISPALYED
FF4C A6 00      DTSR2L LDA A  0,X - GET PATTERN BYTE
FF4E B7 8000     STA A  DISRAM
FF51 08         INX
FF52 5A         DEC B
FF53 26 F7      BNE      DTSR2L
FF55 39         RTS
*  END  *
*
*
FF60          ORG      $FF60
*****
*
*
*      DATTA TABLE
* - ASSOCIATES DATA ENTRY KEY CODES WITH 16 HEX.
* DISPLAY PATTERNS AND THEIR BINARY VALUES.
*

```


			KEY DISP BI.	II	BINARY	I	KEY
			CODE PAT NO.	II	DISP.PAT.	I	CODE
				II		I	
FF60	1B	DATTBL	FCB	\$1B,\$7D,\$00	II	01111101	I 033
FF61	7D						
FF62	00						
FF63	1A		FCB	\$1A,\$05,\$01	II	00000101	I 032
FF64	05						
FF65	01						
FF66	19		FCB	\$19,\$5B,\$02	II	01011011	I 031
FF67	5B						
FF68	02						
FF69	18		FCB	\$18,\$4F,\$03	II	01001111	I 030
FF6A	4F						
FF6B	03						
FF6C	13		FCB	\$13,\$27,\$04	II	00100111	I 023
FF6D	27						
FF6E	04						
FF6F	12		FCB	\$12,\$6E,\$05	II	01101110	I 022
FF70	6E						
FF71	05						
FF72	11		FCB	\$11,\$7E,\$06	II	01111110	I 021
FF73	7E						
FF74	06						
FF75	10		FCB	\$10,\$45,\$07	II	01000101	I 020
FF76	45						
FF77	07						
FF78	0B		FCB	\$0B,\$7F,\$08	II	01111111	I 013
FF79	7F						
FF7A	08						
FF7B	0A		FCB	\$0A,\$67,\$09	II	01100111	I 012
FF7C	67						
FF7D	09						
FF7E	09		FCB	\$09,\$77,\$0A	II	01110111	I 011
FF7F	77						
FF80	0A						
FF81	08		FCB	\$08,\$3E,\$0B	II	00111110	I 010
FF82	3E						
FF83	0B						
FF84	03		FCB	\$03,\$78,\$0C	II	01111000	I 003
FF85	78						
FF86	0C						
FF87	02		FCB	\$02,\$1F,\$0D	II	00011111	I 002
FF88	1F						
FF89	0D						
FF8A	01		FCB	\$01,\$7A,\$0E	II	01111010	I 001
FF8B	7A						
FF8C	0E						
FF8D	00		FCB	\$00,\$72,\$0F	II	01110010	I 000
FF8E	72						
FF8F	0F						


```

*   END   *
*
*
*****
*
*
*   GENERAL INTERRUPT MEMORY ASSIGNMENT
* - INTERRUPT VECTORS CAN BE CHANGED EITHER FROM THE
*   SOURCE CODE THROUGH THE "EQU" STATEMENTS ,
*   OR IN EPROM AT LOCATIONS $FFF8 TO $FFFF
*
*   SWIVEC - THE SWI HANDLER
* - SENDS CONTROL DIRECTLY TO THE SWI ROUTINE
*   SPECIFIED IN THE CALLING SEQUENCE
* - CALLING SEQUENCE:  SWI
*                       FDB (2 BYTES DATA),
*                       (SWI ROUTINE MNEMONIC)
*                       NEXT INSTRUCTION
* - FOR A LIST OF SWI MNEMONICS SEE SWITBL AT $F400
* - CHANGES NO REGISTERS IN MAIN EXCEPT PC WHICH IS
*   SET TO THE LOCATION OF "NEXT INSTRUCTION"
* - RTI CAN BE USE TO CONTINUE IN MAIN
*
FF90 30          SWIVEC TSX          ADD. OF TOP OF STACK
FF91 A6 05          LDA A   5,X      INCREMENT RETURN ADD.
FF93 E6 06          LDA B   6,X      BY 4 SO WE CAN USE RTI
FF95 CB 04          ADD B   #$04
FF97 89 00          ADC A   #$00
FF99 A7 05          STA A   5,X
FF9B E7 06          STA B   6,X
FF9D EE 05          LDX      5,X      X=NEW RETURN ADDRESS
FF9F 09            DEX
FFA0 09            DEX              X=LOCATION OF SWI NEM.
FFA1 EE 00          LDX      0,X      X=LOC. OF SWI ROUT.ADD.
FFA3 EE 00          LDX      0,X      X=SWI ROUTINE ADDRESS
FFA5 6E 00          JMP      0,X      JUMP TO SWI ROUTINE
*           70USEC.
**   END   **
*
*
FFBB          ORG      $FFBB
*****
*
*
*   JUMP SUBROUTINE LINKAGE TABLE
*
* - ALLOWS SUBROUTINES TO CHANGE LOCATION
*   WITHIN A ROM
* - ALLOWS EACH ROM TO BE ASSEMBLED INDIPENDANTLY
*
FFBB 7E FF26          JMP      WRITE0
FFBE 7E FDB9          JMP      FLSH
FFC1 7E FEB2          JMP      READ2
FFC4 7E FF16          JMP      WRITE2

```



```

FFC7 7E FEC1      JMP    RDWL
FFCA 7E FD91 VDSPSN JMP    DSPSN
FFCD 7E FDA3 VDSPDP JMP    DSPDP
FFD0 7E FE42 VCLDSP JMP    CLDISP
FFD3 7E FF2E VDTSR1 JMP    DTSR1
FFD6 7E FF47 VDTSR2 JMP    DTSR2
FFD9 7E FE66 VDSBI  JMP    DSBISR
FFDC 7E FED8 VBIDS  JMP    BIDSSR
FFDF 7E FDE2 VDPAXB JMP    DSPXAB
FFE2 7E FE7B VDSTBI JMP    DSTOBI
FFE5 7E FEEA VBITDS JMP    BITODS
FFE8 7E FE91 VREAD4 JMP    READ4
FFEB 7E FEFB VWRIT4 JMP    WRITE4
FFEE 7E FDF6 VCNP1  JMP    CNPIA1
FFF1 7E FE2E VDEL2  JMP    DELAY2
FFF4 7E FC1F VRMRST JMP    RM1RST

```

*

* INTERRUPT VECTORS

*

*

```

F800    IRQVEC EQU    $F800
FBF0    NMIVEC EQU    TRAP
FC00    RSTVEC EQU    RESET

```

*

*

```

FFF8          ORG    $FFF8
FFF8 F800      FDB    IRQVEC,SWIVEC,NMIVEC,RSTVEC
FFFA FF90
FFFC FBF0
FFFE FC00

```

*

END

ROM2

M68SAM is the property of Motorola Spd, Inc.
Copyright 1974 by Motorola Inc.

Motorola M6800 Cross Assembler, Release 1.1

NAM ROM2 MONITOR

```

*
* SOURCE FILE LABELS
*
* LATCH LABELS. ( SN74LS273 )
*
8800 TRH EQU $8800 - ADDRESS TRAP LABELS
8801 TRL EQU $8801
8C00 LCON EQU $8C00 - CONTROL LATCH
*
*
* TIMER ADDRESSES ( MC6840 )
*
7C01 TSTATR EQU $7C01 - STATUS REGISTER; READ ONLY.
*
*
* ACIA ADDRESSES ( MC6850 )
*
8408 ACI1SR EQU $8408 - STATUS REGISTER
*
*
* PROGRAMMABLE KEYBOARD DISPLAY INTERFACE (I8279-5)
*
8001 PKDICR EQU $8001 - CONTROL REGISTER;WRITE ONLY
8001 PKDISR EQU $8001 - STATUS REGISTER;READ ONLY
8000 FIFO EQU $8000 - KEYBOARD INPUT RAM;READ ONLY
8000 DISRAM EQU $8000 - DISPLAY RAM; READ/WRITE
*
*
* MONITOR GLOBAL VARIABLES AND FLAGS
*
0000 READEN EQU $0000 - READ ENABLE
0001 STFLG EQU $0001 - STOP FLAG
0002 ENTRYC EQU $0002 - ENTRY COUNTER
0004 ENTRY1 EQU $0004 - ENTRIES VIA ENTR FUNCTION
0005 ENTRY2 EQU $0005
0006 ENTRY3 EQU $0006
0007 ENTRY4 EQU $0007
0008 ENTRY5 EQU $0008
0009 ENTRY6 EQU $0009
000A ENTRY7 EQU $000A
000B ENTRY8 EQU $000B
000C LISTC EQU $000C
000D IXRV EQU $000D,E -X REG. STORAGE,MAIN LINES
000F GOTOFE EQU $000F - GO TO FUNCTION ENABLE FLAG
0010 MONFLG EQU $0010 - IND. M.L. STOPED BY MONITOR

```



```

0011     SECFLG EQU $0011 - IND. 2ND KEYBRD. FTN. SEL.
0012     LEDNO  EQU $0012 - LED TO BE WRITTEN INTO NEXT
          *          ( FIRST LED# = 0 )
          *
          *

```

```

          *      SUBROUTINE LABLES
          *

```

```

FFD0     CLDISP EQU $FFD0
FFD3     DTSR1  EQU $FFD3
FFD6     DTSR2  EQU $FFD6
FFD9     DSBISR EQU $FFD9
FFDC     BIDSSR EQU $FFDC
FFDF     DSPXAB EQU $FFDF
FFE2     DSTOBI EQU $FFE2
FFE5     BITODS EQU $FFE5
FFE8     READ4  EQU $FFE8
FFEB     WRITE4 EQU $FFEB
FFEE     CNPIA1 EQU $FFEE
FFF1     DELAY2 EQU $FFF1
F7E0     BITOAS EQU $F7E0
F7E3     TX1CHS EQU $F7E3
F7EC     ACIH   EQU $F7EC
FF60     DATTBL EQU $FF60
F3D3     PTMH   EQU $F3D3

```

```

          *
          *      LABLES FOR BMC INTERRUPTS
          *

```

```

B7E0     BDIH   EQU $B7E0
B7E3     BCIH   EQU $B7E3
AC04     BICS   EQU $AC04 -BUBBLE INT. CONTROL/STATUS

```

```

          *
          *      LABLES FOR KEYBOARD ROUTINES
          *

```

```

F7E6     ERROR  EQU $F7E6
BFE2     SSPL   EQU $BFE2
E7E2     FLAG   EQU $E7E2
E7D9     BDB    EQU $E7D9
BFD9     SAVE   EQU $BFD9
BFDC     TX     EQU $BFDC
E7E8     CAL    EQU $E7E8
BFD6     SEL    EQU $BFD6

```

```

          *
          *      LINKAGE TABLE ENTRIES FOR SWI  ROUTINES
          *

```

```

F400     IERR   EQU $F400
F402     ISTX1  EQU $F402
F404     ISTX2  EQU $F404
F406     ISABX  EQU $F406
F408     ILABX  EQU $F408

```

```

          *
          *****

```



```

***
F800          ORG    $F800
***
*
*
*   INPUT , THE IRQ POLLING ROUTINE
* - CHECKS TIMER,PKDI,BMC,AND ACIA STATUS REGISTERS
*   FOR THE SOURCE OF THE INTERRUPT
* - TIMER HAS HIGHEST PRIORITY,ACIA'S NEXT,THEN PKDI
* - IF NEITHER OF THESE CAUSED THE INTERRUPT,
*   "INT ERR" IS DISPLAYED
F800 B6 7C01 INPUT LDA A TSTATR - READ TIMER STATUS
F803 2A 03          BPL   IRQS00 - JUMP ON TIMER INT.
F805 7E F3D3        JMP   PTMH
*
*
F808 B6 8408 IRQS00 LDA A ACI1SR - TEST STATUS OF ACIA
F80B 2A 03          BPL   IRQS1  - JUMP ON ACIA INT.
F80D 7E F7EC        JMP   ACIH
F810 F6 8001 IRQS1  LDA B PKDISR - READ PKDI STATUS
F813 C5 07          BIT B #$07  - CHECK FOR PKDI INT.
F815 26 18          BNE   IRQS2
F817 B6 AC04        LDA A BICS   CHECK FOR BMC DATA INT.
F81A 2A 03          BPL   IRQS3  SKIP IF NOT
F81C 7E B7E0        JMP   BDIH   GO TO BMC DATA HANDLER
F81F 48             IRQS3 ASL A   CHECK FOR BMC COMMAND INT.
F820 2A 03          BPL   IRQS4  SKIP IF NOT
F822 7E B7E3        JMP   BCIH   GO TO BMC HANDLER
F825 BD FBC4 IRQS4  JSR   PKDIRS
F828 CE FBBC        LDX   #INTERR
F82B BD FFD6        JSR   DTSR2
F82E 3B            RTI
F82F C5 00          IRQS2 BIT B  #$00 -TEST FIFOSR
F831 27 08          BEQ   IRQS5  - NO ERROR
F833 C6 C2          ERROR3 LDA B  #$C2 - CLEAR FIFO STATUS
F835 F7 8001        STA B  PKDICR
F838 7E F7E6        JMP   ERROR
*
*
F83B C4 07          IRQS5 AND B  #$07 - # BYTES IN FIFO
F83D 01            NOP
F83E 01            NOP
F83F 96 00          LDA A  READEN - TEST READ ENABLE
F841 26 06          BNE   KEYIN
F843 C6 C2          LDA B  #%11000010 - CLEAR FIFO
F845 F7 8001        STA B  PKDICR
F848 3B            RTI
*
*
F849 86 40          KEYIN LDA A  #$40 - IRQ MASK SET
F84B B7 8001        STA A  PKDICR
F84E B6 8000 KYINL1 LDA A  FIFO - READ LAST VALUE
F851 5A            DEC B      IN FIFO
F852 26 FA          BNE   KYINL1
F854 36            PSH A      ** - STORE KEYCODE TWICE
F855 36            PSH A      ***
F856 84 38          AND A  #$38

```



```

F858 81 20          CMP A  #$20  - TEST ROW BITS
F85A 2D 03          BLT      KYINS1
F85C 7E F7E6        JMP      ERROR - INVALID KEY ENTRY
F85F 81 10          KYINS1 CMP A  #$10
F861 32             PUL A      *** - RESTORE
F862 2C 27          BGE      KYINS2 - FTN. OR DATA KEY?
      *             *
F864 84 07          AND A  #$07  - DEBUG OR DATA KEY?
F866 81 04          CMP A  #$04
F868 32             PUL A      ** - RESTORE KEYCODE
F869 2D 3F          BLT      KYINS3 - BRANCH IF DATA KEY
F86B 4D             TST A      - DEBUG KEY, BRANCH IF
F86C 2B 08          BMI      KYINS4 ENABLED, (CNTL/STB=1).
F86E 81 0C          CMP A  #$0C
F870 26 03          BNE      KYIRTI
F872 7E F999        JMP      GOTOF   GO TO FTN. ENABLED?
F875 3B             KYIRTI RTI
      *             *
F876 C6 08          KYINS4 LDA B  #$08  - B = # OF DEBUG KEYS
F878 CE FB70        LDX      #DBTBL
F87B A1 00          DBL1    CMP A  0,X
F87D 27 08          BEQ      DBFND
F87F 08             INX
F880 08             INX
F881 08             INX
F882 5A             DEC B
F883 26 F6          BNE      DBL1
F885 20 AC          ERROR2 BRA      ERROR3
F887 EE 01          DBFND  LDX      1,X  - GET DEBUG FTN. ADD.
F889 6E 00          JMP      0,X  - GO TO DEBUG FUNCTION
      *             *
F88B 84 07          KYINS2 AND A  #$07  FUNCTION OR DATA KEY?
F88D 81 04          CMP A  #$04
F88F 32             PUL A      ** - RESTORE KEYCODE
F890 2D 18          BLT      KYINS3 BRANCH IF A DATA KEY
F892 84 3F          AND A  #$3F  - FTN. KEY, INDEPENDANT
F894 CE FB88        LDX      #FTNTBL OF CNTL/STB & SHIFT
F897 C6 08          LDA B  #$08  - B = # OF FTN. KEYS
F899 A1 00          FTNL1  CMP A  0,X
F89B 27 09          BEQ      FTNFND
F89D 08             INX
F89E 08             INX
F89F 08             INX
F8A0 5A             DEC B
F8A1 26 F6          BNE      FTNL1
F8A3 7E F7E6        JMP      ERROR
F8A6 EE 01          FTNFND LDX      1,X  - GET FUNCTION ADDRESS
F8A8 6E 00          JMP      0,X  - JUMP TO FUNCTION
      *             *
F8AA 84 3F          KYINS3 AND A  #$3F  DATA KEYS INDEPENDANT
F8AC CE FF60        LDX      #DATTLBL OF CNTL/STB & SHIFT
F8AF C6 10          LDA B  #$10  - B = # OF DATA KEYS
F8B1 A1 00          DATAL1 CMP A  0,X
F8B3 27 09          BEQ      DATFND

```



```

F8B5 08      INX
F8B6 08      INX
F8B7 08      INX
F8B8 5A      DEC B
F8B9 26 F6    BNE     DATAL1
F8BB 7E F7E6  JMP     ERROR
F8BE A6 01    DATFND LDA A 1,X  - A = DISPLAY PATTERN
F8C0 B7 8000  STA A  DISRAM  DIPLAY & STORE DIGIT
F8C3 7C 0012  INC     LEDNO  # OF LED WRITTEN NEXT
F8C6 3B      RTI

```

* *

*

* STOP - DEBUG SUBROUTINE

* - STOPS MAIN OR NMI ROUTINES, (IF CLI)

* - DISPLAYS NEXT OP CODE AND ADDRESS

* - WAITS FOR NEXT KEY ENTRY, (IRQ)

* - ALL REGISTERS CAN BE SCANNED

* - IF STRT IS DEPRESSED, PROGRAM CONTINUES AS BEFORE

* - TO CHANGE OP CODE OR A BYTE OF DATA DEPRESS TWO #

* KEYS AND ONE OF "STOP,STRT,LIST,SCAN,OR FWD"

*

```

F8C7 BD F952 STOP JSR     ALT    - ALTER OP CODE
F8CA BD F93F      JSR     STADJ
F8CD BD FFD0      JSR     CLDISP
F8D0 30           TSX
F8D1 A6 05        LDA A 5,X    - A = (PCRETURN)H
F8D3 E6 06        LDA B 6,X    - B = (PCRETURN)L
F8D5 BD FFEB      JSR     WRITE4 - DISPLAY RETURN ADD.
F8D8 30           TSX
F8D9 EE 05        LDX     5,X    - RETURN ADDRESS IN X
F8DB A6 00        LDA A 0,X    - DATA AT RET. ADD.
F8DD BD FFE5      JSR     BITODS - DISPLAY DATA
F8E0 B7 8000      STA A  DISRAM
F8E3 F7 8000      STA B  DISRAM

```

* *

```

F8E6 86 01        LDA A #$01
F8E8 97 01        STA A  STFLG  INDICATE RTI ON STACK
F8EA 7F 000C      CLR     LISTC  - USED TO EXIT "SCAN"
F8ED 7C 0010      INC     MONFLG - IND. MAIN STOPED
F8F0 0E           CLI          - ALLOW INTERRUPTS
F8F1 3E           STOPL WAI     - STOP
F8F2 20 FD        BRA     STOPL - WAIT

```

* END *

*

*

* BRKP - DEBUG SUBROUTINE

* - "BREAK POINT" SETS THE ADDRESS TRAP AT THE

* PREVIOUSLY ENTERED ADDRESS.

* - WHEN THE OP CODE AT THIS ADDRESS IS EXECUTED,

* THE NMI ROUTINE IS ENTERED, WHICH PREPARES FOR

* AND ENTERS THE STOP ROUTINE.

* - CAN BE USED TO STOP PROGRAM FLOW

* - ENABLES THE ADDRESS TRAP

* - IF THE BREAK POINT ADDRESS WAS NOT PROPERLY KEYED


```

*      IN, ERROR IS FLASHED AND THE STOP IS ENTERED.
*
F8F4 96 11      BRKP      LDA A      SECFLG      - SKIP IF 2ND FTN.
F8F6 27 06                      BEQ          BRKPS0      NOT SELECTED
*
F8F8 BD FAE1          JSR          SCNDSR
F8FB 7E E7D9          JMP          BDB      - GO TO 2ND FUNCTION
*
F8FE BD F93F BRKPS0 JSR          STADJ
F901 4F                      CLR A
F902 BD FFE8          JSR          READ4
F905 24 0C                      BCC          BRKPS1 - IF ENTRY O.K., SKIP
F907 3F                      SWI
F908 0000          FDB          0, IERR
F90A F400
F90C BD FFF1          JSR          DELAY2
F90F 01F9          FDB          $01F9
F911 20 B4          BRA          STOP
F913 B7 8800 BRKPS1 STA A      TRH      - STORE TRAP ADDRESS
F916 F7 8801          STA B      TRL
F919 86 01          LDA A      #$01
F91B B7 8C00          STA A      LCON      - ENABLE ADDRESS TRAP
F91E 97 01          STA A      STFLG      - SET STOP FLAG
F920 86 94          LDA A      #$94      WRITE "TRS"--TRAP SET
F922 B7 8001          STA A      PKDICR
F925 4F          CLR A
F926 C6 45          LDA B      #%01000101
F928 B7 8000          STA A      DISRAM
F92B F7 8000          STA B      DISRAM
F92E 86 12          LDA A      #%00010010
F930 C6 EE          LDA B      #%11101110
F932 B7 8000          STA A      DISRAM
F935 F7 8000          STA B      DISRAM
F938 7C 0010          INC          MONFLG      - IND. MAIN STOPPED
F93B 0E          CLI
F93C 3E          BRKPL WAI
F93D 20 FD          BRA          BRKPL
*
*
*      DEBUG SUBROUTINE "STACK ADJUST"
* - IF THE MPU WAS IN THE WAI MODE AND STFLG WAS SET
* THEN THIS PROGRAM REMOVES THE RETURN FROM INT.
* VTRS, (POINTING TO THE DEBUG PROG.), FROM THE STACK
* SO THE NEXT RTI WILL SEND CONTROL BACK TO "MAIN"
* - WHILE IN MAIN STFLG = $00
*
F93F 96 01      STADJ      LDA A      STFLG      - TEST STFLG
F941 26 01                      BNE          STADJS
F943 39                      RTS
F944 30      STADJS      TSX
F945 EE 00          LDX          0, X      - X = RETURN ADDRESS
F947 86 09          LDA A      #$09      - INC. STACK PTR BY 9
F949 31      STADJL      INS
F94A 4A          DEC A

```



```

F94B 26 FC          BNE      STADJL
F94D 7F 0001        CLR      STFLG
F950 6E 00          JMP      0,X
*   END   *
*
*   DEBUG SUBROUTINE "ALTER"
* - USED TO ALTER ONE BYTE OF OP CODE OF DATA
* - FIRST TESTS TO SEE IF AN ALTERATION WAS REQUESTED
*   BY CHECKING IF TWO EXTRA DATA BYTES WERE ADDED
*   TO THE STOP MODE FORMAT IN THE DISPLAY.
* - DISPLAY RAM BYTES 6,7 BOTH MUST B PROPER DISPLAY
*   CODE BEFORE SUBROUTINE WILL ALTER.
*
F952 96 01      ALT      LDA A  STFLG
F954 27 1D          BEQ      ALTRTS  MUST BE IN STOP MODE
F956 96 0C          LDA A  LISTC
F958 26 19          BNE      ALTRTS  MUST NOT BE IN LIST
F95A 86 76          LDA A  #$76      - PREPARE TO READ
F95C B7 8001        STA A  PKDICR    STARTING AT BYTE 6
F95F B6 8000        LDA A  DISRAM    - READ BYTE 6
F962 27 0F          BEQ      ALTRTS
F964 F6 8000        LDA B  DISRAM
F967 27 0A          BEQ      ALTRTS
F969 BD FFE2        JSR      DSTOBI  - NEW BINARY DATA
F96C 25 05          BCS      ALTRTS
F96E 30            TSX      - 2ND IRQ FROM MAIN
F96F EE 0E          LDX      14,X    - ADD. OF OPCODE/DATA
F971 A7 00          STA A  0,X      STORE DATA AT ADD.
F973 39            ALTRTS RTS      IN MAIN
*   END   *
*
*
*   "START" - DEBUG SUBROUTINE
* - "START" STARTS THE MPU RUNNING FROM WHERE IT WAS
*   STOPPED, (USUALLY MAIN.)
* - IF OP CODE OR DATA IS TO BE CHANGED BY INSERTING
*   2 DIGITS AFTER THE STOP MODE FORMAT,ALT WILL DO
* - CLEARS STFLG,CLEARS DISPLAY,READY TO WRITE DISP.
* - ALT MODE MUST BE ENTERED AFTER STOP,DISC,GOTO,
*   TRAN, OR FWD      IE.) WHILE IN A WAI.
*
F974 7F 0010  STRT      CLR      MONFLG - IND. MAIN RUNNING
F977 96 01          LDA A  STFLG      (MONITOR INACTIVE)
F979 27 0C          BEQ      STTRTI  - RTI IF NOT IN STOP
F97B BD F952        JSR      ALT      - ALTER OP CODE/DATA
F97E BD F93F        JSR      STADJ
F981 BD FFD0        JSR      CLDISP
F984 7F 000C        CLR      LISTC
F987 3B            STTRTI RTI      - BACK TO MAIN
*   END   *
*
*
*   GOTO - DEBUG SUBROUTINE
* - SETS PROGRAM COUNTER TO JUST ENTERED ADDRESS

```



```

* - ENTER:   CLR   - FUNCTION KEY
*             DATA - ADD. ENTERED INTO DISPLAY
*             GOTO  - FUNCTION KEY
* - ENTERS STOP MODE TO DISPLAY PC & OPCODE OR DATA
* - CAN BE USED TO STOP PROGRAM FLOW
F988 BD F93F GOTO   JSR   STADJ
F98B 4F          CLR A
F98C BD FFE8     JSR   READ4 - ADDRESS IN A,B
F98F 25 05      BCS   GOTOS1 - IF ERROR,SKIP
F991 30         TSX
F992 A7 05      STA A 5,X   TRANSFER ADD.TO RETURN
F994 E7 06      STA B 6,X   VECTOR IN STACK
F996 7E F8C7 GOTOS1 JMP   STOP
**   END   **
*
*   GOTOF  DISABLED DEBUG VERSION OF "GOTO"
*   - SENDS CONTROL TO THE 16 BIT VECTOR
*   IN THE DISPLAY RAM
*   - DOES NOT STOP THE MPU
*   - ENTERED VIA A PKDI INTERRUPT
F999 96 0F     GOTOF LDA A GOTOF
F99B 27 01     BEQ   GOTOF5 SKIP IF GOTOF ENABLED
F99D 3B        RTI
F99E 4F        GOTOF5 CLR A
F99F BD FFE8     JSR   READ4  STORE ADDRESS IN A,B
F9A2 30        TSX
F9A3 A7 05      STA A 5,X   STORE ADD. IN RETURN
F9A5 E7 06      STA B 6,X   VECTOR ON STACK
F9A7 7F 0010    CLR   MONFLG IND. MAIN RUNNING
F9AA 7E FAC4    JMP   CLR1   CLR DISP.,FIFO, RTI
**   END   **
*
*   ENTR  - GENERAL FUNCTION
*   - USED TO ENTER 16 BYTE DATA FROM DISPLAY
*   - CONVERTS THE FIRST 4 LED DISP. CONTENTS & STORES
*   THEM AS 16 BITS AT ENTRY1,ENTRY2,ENTRY3,AND ENTRY4
*   - ENTRYC CONTAINS # OF PREVIOUS ENTRY BYTES
*   - DOES NOT STOP PROGRAM FLOW, USES RTI
*
F9AD 96 03     ENTR   LDA A  ENTRYC+1
F9AF 81 04     CMP A  #$04
F9B1 2D 06     BLT   ENTR51 - IF 2 NUM.ALREADY
F9B3 7F 0002    CLR   ENTRYC  ENTERED, OVERWRITE
F9B6 7F 0003    CLR   ENTRYC+1
F9B9 4F        ENTR51 CLR A
F9BA BD FFE8     JSR   READ4  - STORE DATA IN A,B
F9BD DE 02      LDX   ENTRYC  X=OFFSET FORM ENTRY1
F9BF A7 04      STA A  ENTRY1,X
F9C1 E7 05      STA B  ENTRY1+1,X
F9C3 08        INX
F9C4 08        INX
F9C5 DF 02      STX   ENTRYC
F9C7 A6 02      LDA A  ENTRY1-2,X
F9C9 E6 03      LDA B  ENTRY1-1,X

```



```

F9CB BD FFEB      JSR      WRITE4
F9CE 86 90        LDA A    #%10010000
F9D0 B7 8001      STA A    PKDICR
F9D3 3B           RTI
**   END   **
*
*
*   LIST      - DEBUG SUBROUTINE
* - ENTERED VIA AN INTERRUPT FROM KEYBOARD
* - USED TO DISPLAY REGISTERS AFTER MPU WAS STOPPED
*   IE) AFTER STOP,DISC,GOTO,TRANS,OR FWD
* - DOES NOT STOP MPU
* - DISPLAYS ACC. A
* - USED WITH SCAN TO DISPLAY & ALTER REGISTERS
* - CAN ALTER THE DISPLAYED OPCODE OR DATA
*
F9D4 96 01      LIST      LDA A    STFLG   - IN STOP MODE?
F9D6 26 01              BNE      LISTS1 - BRANCH IF SO
F9D8 3B              RTI              - EXIT IF NOT
F9D9 BD F952 LISTS1 JSR      ALT      - REPLACE OPCODE/DATA
F9DC 86 01              LDA A    #$01
F9DE 97 0C              STA A    LISTC   - INIT. LIST COUNTER
*
**   CONTROL FLOWS INTO SCAN
*
*   SCAN      - DEBUG SUBROUTINE
* - SYSTEM MUST BE IN STOP MODE OR WE RTI
*   MODE1      - USED AFTER LIST TO DISPLAY OTHER MPU REG.
*               (LISTC>0, LISTC DETERMINES WHICH REG. IS
*               DISPLAYED AND WHICH IS ALTERED )
*               - ALTERS A REGISTER, DISPLAYS NEXT REG.,
*               INCREMENTS LISTC, THEN EXECUTES WAI
*               - DISPLAY FORMAT: IIIIRRBB
*               I=REGISTER IDENTIFIER
*               R=DATA TO BE PLACED IN REGISTER
*               B=BLANK
*   MODE2      - USED ONCE IN STOP MODE TO DISP.AND CHANGE
*               SUCCESSIVE OPCODES OR DATA AT LOCATION
*               POINTED TO BY THE PROGRAM COUNTER IN MAIN.
*               - ALTERS DATA, DISPLAYS DATA IN NEXT LOC.
*               INCREMENTS PROGRAM COUNTER IN MAIN.
*               - DISPLAY FORMAT: LLLLDCC
*               L=LOC. OF DATA = PROG. CNTR. VALUE IN MAIN
*               D=DATA IN LOCATION
*               C=DATA ENTERED TO BE STORED IN LOCATION
*
F9E0 96 0C      SCAN      LDA A    LISTC   - TEST IF IN LIST
F9E2 27 54              BEQ      SMODE2 - BRANCH IF NOT
*               - IN LIST MODE, GET LIST COUNTER
F9E4 CE F9F2              LDX      #SCANT - X = LOC. OF TABLE
F9E7 4A          SCANL1 DEC A              - GET ROUTINE ADDRESS
F9E8 27 04              BEQ      SCANS1
F9EA 08              INX
F9EB 08              INX

```



```

F9EC 20 F9      BRA      SCANL1
F9EE EE 00      SCANS1 LDX      0,X      - GET RIGHT ROUTINE
F9F0 6E 00      JMP      0,X      - JUMP TO ROUTINE
*
*
F9F2 F9FC      SCANT  FDB  DISPA,DISPB,DISPC,DISPI,DISPPC
F9F4 F9FE
F9F6 FA07
F9F8 FA10
F9FA FA19
*
*
F9FC 20 5F      DISPA  BRA      DSPR      - BRANCH TO ROUTINE
F9FE 8D 4E      DISPB  BSR      SCRDDS    - READ VALUE OF A
FA00 25 03      BCS      DISPBS
FA02 30      TSX      - CHANGE ONLY IF O.K.
FA03 A7 09      STA  A  9,X
FA05 20 56      DISPBS BRA      DSPR      - DISPLAY B FROM MAIN
*
FA07 8D 45      DISPC  BSR      SCRDDS    - READ DISPLAYED B
FA09 25 03      BCS      DISPCS
FA0B 30      TSX      CHANGE ONLY IF NO ERROR
FA0C A7 08      STA  A  8,X
FA0E 20 4D      DISPCS BRA      DSPR      - DISPLAY C FROM MAIN
*
FA10 8D 3C      DISPI  BSR      SCRDDS    - READ DISPLAYED C
FA12 25 03      BCS      DISPIS
FA14 30      TSX      - CNG. ONLY IF NO ERROR
FA15 A7 07      STA  A  7,X
FA17 20 44      DISPIS BRA      DSPR      - DISPLAY I FORM MAIN
*
FA19 8D 33      DISPPC BSR      SCRDDS    - READ DISPLAYED I
FA1B 25 12      BCS      DSPPCS
FA1D 36      PSH  A  **      - PUSH IXRH
FA1E B6 8000      LDA  A  DISRAM
FA21 F6 8000      LDA  B  DISRAM    - GET IXRL FROM DISP.
FA24 BD FFE2      JSR      DSTOBI
FA27 33      PUL  B  **      - PULL IXRH
FA28 25 05      BCS      DSPPCS    - BRANCH IF ERROR
FA2A 30      TSX
FA2B E7 0A      STA  B  10,X    - STORE IXRH
FA2D A7 0B      STA  A  11,X    - STORE IXRL
*
FA2F 7F 000C      DSPPCS CLR      LISTC    SET LISTC,NOT IN LIST
FA32 BD F93F      JSR      STADJ    - ADJUST STACK
FA35 7E F8C7      JMP      STOP      - JUMP TO STOP MPU
*
*
FA38 BD F952      SMODE2 JSR      ALT      - ALTER OP CODE/DATA
FA3B BD F93F      JSR      STADJ
FA3E 30      TSX
FA3F A6 05      LDA  A  5,X      INC. PROG. CTR. IN MAIN
FA41 E6 06      LDA  B  6,X
FA43 CB 01      ADD  B  #$01

```



```

FA45 89 00      ADC A    #$00
FA47 A7 05      STA A    5,X
FA49 E7 06      STA B    6,X
FA4B 7E F8C7    JMP      STOP
**  END  **
*
*
FA4E 86 74      SCRDDS  LDA A    #$74      "SCAN DISPLAY READ"
FA50 B7 8001    STA A    PKDICR
FA53 B6 8000    LDA A    DISRAM READ STARTING AT LED4
FA56 F6 8000    LDA B    DISRAM - READ 2 DIGITS
FA59 BD FFE2    JSR      DSTOBI - CONVERT TO BINARY
FA5C 39        RTS
**  END  **
*
*
FA5D 86 90      DSPR    LDA A    #$90      "DISPLAY REGISTER"
FA5F B7 8001    STA A    PKDICR WRITE IN DISP.BYTE 0
FA62 96 0C      LDA A    LISTC LISTC DETERMINES REG.
FA64 CE FBA0    LDX      #DSPRTB
FA67 4A        DSPRL1  DEC A    - FIND REG.I.D.PATTERN
FA68 27 09      BEQ      DSPRFD
FA6A 08        INX      ; FCB LISTC VALUE      0,X
FA6B 08        INX      -                      1,X
FA6C 08        INX      I                      2,X
FA6D 08        INX      > FCB 4 BYTE PATTERN 3,X
FA6E 08        INX      -                      4,X
FA6F 08        INX      ; FDB BRANCH VECTOR 5,X
FA70 08        INX      -                      6,X
FA71 20 F4      BRA      DSPRL1
FA73 A6 01      DSPRFD  LDA A    1,X      - DISPLAY REG. I.D.
FA75 B7 8000    STA A    DISRAM
FA78 A6 02      LDA A    2,X
FA7A B7 8000    STA A    DISRAM
FA7D A6 03      LDA A    3,X
FA7F B7 8000    STA A    DISRAM
FA82 A6 04      LDA A    4,X
FA84 B7 8000    STA A    DISRAM
FA87 EE 05      LDX      5,X      - GET JUMP VECTOR
FA89 6E 00      JMP      0,X      - JUMP TO DISP. REG.
*
*
FA8B 30        OUTA    TSX
FA8C A6 09      LDA A    9,X      - LOAD "A" (FROM MAIN)
FA8E 20 19      BRA      DSPREX - DISPLAY REGISTER
FA90 30        OUTB    TSX
FA91 A6 08      LDA A    8,X      - LOAD "B"
FA93 20 14      BRA      DSPREX
FA95 30        OUTC    TSX
FA96 A6 07      LDA A    7,X      - LOAD "CCR"
FA98 20 0F      BRA      DSPREX
FA9A 30        OUTI    TSX
FA9B A6 0A      LDA A    10,X     - LOAD "IH"
FA9D BD FFE5    JSR      BITODS

```



```

FAA0 B7 8000      STA A   DISRAM   - DISPLAY IH
FAA3 F7 8000      STA B   DISRAM
FAA6 30          TSX
FAA7 A6 0B        LDA A   11,X      - LOAD "IL"
FAA9 BD FFE5 DSPREX JSR     BITODS  CNG. BINARY TO DISP.
FAAC B7 8000      STA A   DISRAM   - DISP. REG. VALUE
FAAF F7 8000      STA B   DISRAM
FAB2 7C 000C      INC     LISTC
FAB5 86 94        LDA A   #$94     - READY TO WRITE OVER
FAB7 B7 8001      STA A   PKDICR   THE DISPLAYED VALUE
FABA BD F93F      JSR     STADJ
FABD 7C 0001      INC     STFLG    - INDICATE STOPPED
FAC0 0E          CLI              - ENABLE SYSTEM INT.
FAC1 3E          SCNWAI WAI        - WAIT FOR DATA
FAC2 20 FD        BRA     SCNWAI

```

```

*** END ***

```

```

*

```

```

*

```

```

* CLR1 - "CLEAR", KEYBOARD ACTIVATED FUNCTION
* - RECONFIGURES THE PKDI
* - CLEARS DISPLAY RAM AND KEYBOARD FIFO
* - CLEARS DATA ENTRY REGISTERS, ENTRYI
* - ALLOWS THE DISPLAY RAM TO BE WRITTEN INTO
*   STARTING AT BYTE 0
*

```

```

FAC4 96 11 CLR1 LDA A SECFLG
FAC6 27 06 BEQ   CLR1S1 - SKIP IF CLEAR

```

```

*

```

```

FAC8 BD FAE1 JSR SCNDSR
FACB 7E F9AD JMP  ENTR

```

```

*

```

```

FACE BD FBC4 CLR1S1 JSR PKDIRS - RESET PKDI
FAD1 CE 0002 LDX   #ENTRYC
FAD4 6F 00 CLR1L1 CLR  0,X     - CLEAR ENTRY LOC.
FAD6 08      INX
FAD7 8C 000D CPX   #LISTC+1
FADA 26 F8   BNE   CLR1L1
FADC 3B      RTI

```

```

** END **

```

```

FADD SCND EQU * SECOND

```

```

*

```

```

* - A GENERAL FUNCTION, ( KEYBOARD ACTIVATED )
* - USED TO SELECT THE LOWER FUNCTION ON DUAL PURPOSE
*   FUNCTION KEYS
* - IF SECFLG IS CLEAR, IT IS SET AND A DECIMAL POINT
*   IS DISPLAYED IN LED # 6
* - IF SECFLG IS SET, IT IS CLEARED AND THE DECIMAL
*   POINT IS REMOVED FORM LED # 6
*

```

```

FADD BD FAE1 JSR SCNDSR
FAE0 3B      RTI

```

```

** END **

```

```

*

```

```

*

```


FAE1 SCNDSR EQU * SECOND SUBROUTINE

*

* - PERFORMS THE FUNCTION OF "SECOND"

* - CHANGES CCR,A,B

*

```

FAE1 86 67          LDA A  #$67  PREPARE TO READ LED #7
FAE3 B7 8001        STA A  PKDICR
FAE6 F6 8000        LDA B  DISRAM  READ LED #7
FAE9 96 11          LDA A  SECFLG  TEST SECFLG
FAEB 26 07          BNE     SCNDS1  BRANCH IF SET

```

*

```

FAED 7C 0011        INC     SECFLG  CLEAR SO SET IT
FAF0 CA 80          ORA B  #$80     SET D.P. ON LED #7
FAF2 20 05          BRA     SCNDS2

```

*

```

FAF4 7F 0011 SCNDS1 CLR     SECFLG  SET SO CLEAR IT
FAF7 C4 7F          AND B  #$7F     CLEAR D.P.ON LED #7

```

*

```

FAF9 86 87 SCNDS2 LDA A  #$87     WRITE NEW CHARACTER
FAFB B7 8001        STA A  PKDICR  IN LED #7
FAFE F7 8000        STA B  DISRAM
FB01 96 12          LDA A  LEDNO
FB03 84 0F          AND A  #$0F
FB05 8B 90          ADD A  #$90
FB07 B7 8001        STA A  PKDICR  GET READY TO WRITE
FB0A 39             RTS

```

*

** END **

*

*

* TRAP - DEBUG SUBROUTINE

* - ADDRESS VECTORED TO WHEN A NMI IS GENERATED

* FROM THE ADDRESS TRAP

* - ALWAYS ENTERS STOP ROUTINE

* - DISABLES ADDRESS TRAP

*

```

FB0B 7F 8C00 TRAP  CLR     LCON
FB0E 96 01          LDA A  STFLG
FB10 27 06          BEQ     TRAPEX
FB12 BD F93F        JSR     STADJ  - REMOVE RTI VECTORS
FB15 7C 0001        INC     STFLG
FB18 7E F8C7 TRAPEX JMP     STOP

```

** END **

*

*

*

* FWD - DEBUG SUBROUTINE

* - ALLOWS THE NEXT OP. CODE TO BE EXECUTED, THEN

* STOPS CPU FOR DISP. OF NEXT OP.CODE IN STOP MODE.

* - STORES PC IN THE ADD. TRAP AND ENABLES THE TRAP

* - PC = ADDRESS OF NEXT INSTRUCTION TO BE EXECUTED

* - THEN JUMPS TO STRT

* - WHEN OP.CODE AT THAT ADD. IS FETCHED, NMI OCCURRS

* - CAN ALTER OPCODE BEFORE EXECUTION, IF IN RAM

* - CAN BE USED TO ENTER STOP MODE VIA A NMI

*

```

FB1B BD F952 FWD      JSR      ALT      - CHANGE OPCODE
FB1E BD F93F          JSR      STADJ    - ADJUST STACK
FB21 30              TSX
FB22 EE 05          LDX      5,X      - X = PC FROM MAIN
FB24 FF 8800        STX      TRH      TRAP ADD.= PC FROM MAIN
FB27 86 01          LDA A    #$01
FB29 B7 8C00        STA A    LCON    - ENABLE TRAP
FB2C BD FFD0        JSR      CLDISP
FB2F 7F 000C        CLR      LISTC
FB32 3B            RTI              - BACK TO MAIN

```

** END **

*

*

```

* TRAN - "TRANSLATE" - DEBUG SUBROUTINE
* - MOVES DATA FROM ONE LOCATION TO ANOTHER
* - ENTRY FROM KEYBOARD :
*   CLR ( CLR FUNCTION KEY )
*   A0   - ADDRESS OF START LOCATION OF DATA
*   ENTR ( ENTER FUNCTION KEY )
*   A1   - ADDRESS OF LAST LOCATION OF DATA
*   ENTR
*   A2   - ADDRESS OF FIRST LOCATION OF DESTINATION
*   TRAN
* - CLEARS ENTRYC

```

*

```

FB33 96 03      TRAN      LDA A    ENTRYC+1
FB35 81 04          CMP A    #$04
FB37 26 34          BNE      TRANS1 - ENRTYC MUST BE 4
FB39 4F          CLR A
FB3A BD FFE8      JSR      READ4  - DISPLAYED ADDRESS
FB3D 97 08          STA A    ENTRY5
FB3F D7 09          STA B    ENTRY6 - DESTINATION ADDRESS
FB41 97 0A          STA A    ENTRY7  STORED TWICE
FB43 D7 0B          STA B    ENTRY8
FB45 DE 04          LDX      ENTRY1 - SOURCE LOCATION
FB47 A6 00      TRANL 1    LDA A    0,X
FB49 DE 08          LDX      ENTRY5 - DESTINATION
FB4B A7 00          STA A    0,X
FB4D 08          INX
FB4E DF 08          STX      ENTRY5
FB50 DE 04          LDX      ENTRY1 ( SOURCE LOCATION )
FB52 9C 06          CPX      ENTRY3 ( END LOCATION )
FB54 27 05          BEQ      TRANRT - RETURN IF SAME
FB56 08          INX
FB57 DF 04          STX      ENTRY1
FB59 20 EC          BRA      TRANL 1
FB5B BD F93F      TRANRT   JSR      STADJ
FB5E 30          TSX
FB5F 96 0A          LDA A    ENTRY7 - DEST. LOC., (FIRST)
FB61 D6 0B          LDA B    ENTRY8
FB63 A7 05          STA A    5,X      STORED IN PC OF MAIN
FB65 E7 06          STA B    6,X

```



```

FB67 7F 0002      CLR      ENTRYC
FB6A 7F 0003      CLR      ENTRYC+1
FB6D 7E F8C7 TRANS1 JMP      STOP

```

```

***      END      ***

```

```

*

```

```

*****

```

```

*

```

```

*      TABLES

```

```

*

```

```

*      DBTBL  -  "DEBUG TABLE"

```

```

* - ASSOCIATES BEBUG ROUTINE ADDRESSES WITH KEYCODES

```

```

* - FORMAT :      FCB  KEYCODE

```

```

*                  FDB  ROUTINE ADDRESS

```

```

*

```

```

FB70 8F          DBTBL  FCB      $8F
FB71 F8C7        FDB      STOP
FB73 8E          FCB      $8E
FB74 F974        FDB      STRT
FB76 8D          FCB      $8D
FB77 FB1B        FDB      FWD
FB79 8C          FCB      $8C
FB7A F988        FDB      GOTO
FB7C 87          FCB      $87
FB7D FB33        FDB      TRAN
FB7F 86          FCB      $86
FB80 F9D4        FDB      LIST
FB82 85          FCB      $85
FB83 F9E0        FDB      SCAN
FB85 84          FCB      $84
FB86 F8F4        FDB      BRKP

```

```

**      END      **

```

```

*

```

```

*      FTNTBL -  "FUNCTION TABLE"

```

```

* - FUNCTION ROUTINE ADDRESSES MAPPED ONTO KEYCODES

```

```

*

```

```

FB88 1F          FTNTBL FCB      $1F
FB89 FADD        FDB      SCND
FB8B 1E          FCB      $1E
FB8C BFE2        FDB      SSPL
FB8E 1D          FCB      $1D
FB8F E7E2        FDB      FLAG
FB91 1C          FCB      $1C
FB92 FAC4        FDB      CLR1
FB94 17          FCB      $17
FB95 E7E8        FDB      CAL
FB97 16          FCB      $16
FB98 BFDC        FDB      TX
FB9A 15          FCB      $15
FB9B BFD9        FDB      SAVE
FB9D 14          FCB      $14
FB9E BFD6        FDB      SEL

```

```

*

```

```

*      DSPRTB -  "DISPLAY REGISTER TABLE"

```

```

*ASSOCIATES THE VALUE OF LISTC WITH THE REGISTER I.D.

```


*PATTERN AND THE LOCATION OF FURTHER REGISTER DISPLAY
*INSTRUCTIONS.

*FORMAT : FCB LISTC VALUE
* FCB 4 BYTE DISPLAY PATTERN
* FDB BRANCH VECTOR
*

```
FBA0 01           DSPRTB FCB     1
FBA1 77                   FCB     $77,$80,$00,$00
FBA2 80
FBA3 00
FBA4 00
FBA5 FA8B               FDB     OUTA
FBA7 02                   FCB     2
FBA8 3E               FCB     $3E,$80,$00,$00
FBA9 80
FBAA 00
FBAB 00
FBAC FA90               FDB     OUTB
FAE 03                   FCB     3
FBAF 1A               FCB     $1A,$1A,$12,$80
FBB0 1A
FBB1 12
FBB2 80
FBB3 FA95               FDB     OUTC
FBB5 04                   FCB     4
FBB6 05               FCB     $05,$12,$80,$00
FBB7 12
FBB8 80
FBB9 00
FBBA FA9A               FDB     OUTI
```

** END **

*

* INTERR - "INTERRUPT ERROR"

* - DISPLAY PATTERN FOR AN INTERRUPT ERROR MESSAGE

*

```
FBBC 05           INTERR FCB     $05           I
FBBD 16                   FCB     $16           N
FBBE 45                   FCB     $45           T.
FBBF 80                   FCB     $80,0
FBC0 00
FBC1 7A                   FCB     $7A           E
FBC2 12                   FCB     $12,$12       RR
FBC3 12
```

** END **

*

*

```
FBC4     PKDIRS EQU *   PROGRAMMABLE KEYBOARD DISPLAY
*                               INTERFACE RESET
```

*

* - RECONFIGURES THE PKDI (INTEL 8279)

* - CLEARS THE DISPLAY RAM

* - CLEARS THE KEYBOARD FIFO

* - PREPARES TO WRITE TO LED 0

*


```

FBC4 86 00          LDA A  #$00 8 CHAR. DISP.,LEFT ENT.
FBC6 B7 8001        STA A  PKDICR ENCODED SCAN KEYBD.
FBC9 86 3F          LDA A  #$3F  PROG. CLOCK; 32.3KHZ
FBCB B7 8001        STA A  PKDICR
FBCE 7F 0012        CLR    LEDNO  LED 0 WRITTEN TO NEXT
FBD1 7F 0011        CLR    SECFLG  2ND FTN. NOT SEL.
FBD4 86 D3          LDA A  #%11010011  CLEAR DISP. RAM
FBD6 B7 8001        STA A  PKDICR  & FIFO, RESYNCHRONIZE

```

*

```

FBD9 B6 8001 PKDIRL LDA A  PKDISR WAIT UNTIL READY TO
FBDC 2B FB          BMI    PKDIRL WRITE TO DISPLAY RAM
FBDE 86 90          LDA A  #$90    GET READY TO WRITE
FBE0 B7 8001        STA A  PKDICR  TO LED 0
FBE3 39             RTS

```

*

** END **

*

*

```

FBF0                      ORG    $FBF0

```

*

* JUMP DEBUG ROUTINE LINKAGE TABLE

*

```

FBF0 7E FB0B VTRAP  JMP    TRAP
FBF3 7E F8C7 VSTOP  JMP    STOP
FBF6 7E FAE1        JMP    SCNDSR
FBF9 7E FBC4        JMP    PKDIRS

```

END

ROM3

M68SAM is the property of Motorola Spd, Inc.
Copyright 1974 by Motorola Inc.

Motorola M6800 Cross Assembler, Release 1.1

NAM ROM3 ACIA AND SWI ROUTINES

*

* SOURCE FILE LABELS

*

*

*

ACIA ADDRESSES (MC6850)

*

8408 ACI1CR EQU \$8408 - CONTROL REGISTER
8408 ACI1SR EQU \$8408 - STATUS REGISTER
8409 ACI1TX EQU \$8409 - TRANSMIT DATA REGISTER
8409 ACI1RX EQU \$8409 - RECEIVE DATA REGISTER

*

*

*

PROGRAMMABLE KEYBOARD DISPLAY INTERFACE (I8279-5)

*

8001 PKDICR EQU \$8001 - CONTROL REGISTER;WRITE ONLY
8001 PKDISR EQU \$8001 - STATUS REGISTER;READ ONLY
8000 FIFO EQU \$8000 - KEYBOARD INPUT RAM;READ ONLY
8000 DISRAM EQU \$8000 - DISPLAY RAM; READ/WRITE

*

*

*

MONITOR GLOBAL VARIABLES AND FLAGS

*

000D IXRV EQU \$000D,E - X REG.STORAGE, MAIN LINES
0010 MONFLG EQU \$0010 - IND. MAIN STOPED BY MONITOR

*

*

*

SUBROUTINE LABLES

*

FFD0 CLDISP EQU \$FFD0
FFD3 DTSR1 EQU \$FFD3
FFD6 DTSR2 EQU \$FFD6
FFD9 DSBISR EQU \$FFD9
FFDC BIDSSR EQU \$FFDC
FFDF DSPXAB EQU \$FFDF
FFE2 DSTOBI EQU \$FFE2
FFE5 BITODS EQU \$FFE5
FFC1 READ2 EQU \$FFC1
FFC4 WRITE2 EQU \$FFC4
FFE8 READ4 EQU \$FFE8
FFEB WRITE4 EQU \$FFEB
FFEE CNPIA1 EQU \$FFEE
FFF1 DELAY2 EQU \$FFF1
FF60 DATTBL EQU \$FF60
F3EB CKSM EQU \$F3EB
F3EE GTXB EQU \$F3EE
F3E5 TMHB EQU \$F3E5


```

F3F4    MDTI    EQU $F3F4
B7EC    TNDB    EQU $B7EC

```

```

*
```

```

*
```

```

*    SPECIAL JUMP TABLE ENTRIES

```

```

*
```

```

FBF0    TRAP    EQU $FBF0
FBF3    STOP    EQU $FBF3
FFF4    RM1RST EQU $FFF4    - RESET TO ROM1
F3D0    RM4RST EQU $F3D0

```

```

*
```

```

*
```

```

*    VARIABLES TO BE USED ONLY BY ROM3

```

```

*
```

```

0013                                ORG $0013
0013 0001 BBUF1   RMB 1
0014 0001 EBUF1   RMB 1
0015 0001 BUF1SR  RMB 1
0016 0002 BBUF2   RMB 2
0018 0002 BBUF2T  RMB 2
001A 0002 EBUF2   RMB 2
001C 0002 EBUF2T  RMB 2
001E 0001 BUF2SR  RMB 1 BUF2 STAT;$02 FULL,
001F 0001 TX2FLG  RMB 1          $01 MT, $00 NEITHER
0020 0001 TXCMP   RMB 1
0021 0001 NAFLG   RMB 1 - NO ACKNOWLEDGE FLAG
0022 0001 DC1FLG  RMB 1 - IND. FRONT END OF MTS READY
*
0023 0002 TXPTR   RMB 2
0025 0002 RXPTR   RMB 2 -POINTERS TO RX AND TX ROUTINES
0027 0002 TX2CSX  RMB 2 - X STORAGE FOR TX2CHS

```

```

*
```

```

*
```

```

*    OTHER VARIABLES USED BY ROM 3

```

```

*
```

```

0040                                ORG $0040
0040 0002 BSILP   RMB 2
0042 0001 FLGC    RMB 1
0043 0002         RMB 2
0045 0002         RMB 2
0047 0002 TXBLKP  RMB 2 - POINTER TO TXBLK
0049 0001 RXD     RMB 1 - RX DATA

```

```

*
```

```

*
```

```

*
```

```

*    CONSTANTS USED BY ROM3

```

```

*
```

```

0700    BUF1     EQU $700
1000    BUF2     EQU $1000 - LOCATION OF BUFFER #2
1800    BUF2ST   EQU $1800 - STOP FOR BUF2, 2K BUFFER
0490    Y        EQU $0490 - LOC. OF HEX BLOCK TO BE TXED
0560    TXBLK    EQU $0560 - TX BLOCK LOCATION
0650    BSIL     EQU $0650 - BMS INPUT LINE
0690    BSOL     EQU $0690 - BMS OUTPUT LINE

```

```

*
```



```

*****
*
F400          ORG      $F400
*
*          *****
*
*      SWI ROUTINE ADDRESS TABLE
*
* SWI      ROUTINE      COMMENTS
*FUNCTION  ADDRESS
*MNEMONIC
*
F400 F412 IERR      FDB ERROR DISPLAY "ERROR"
F402 F4C3 ISTX1     FDB STX1  TRANSMIT OR STORE BYTE
F404 F5B4 ISTX2     FDB STX2  STORE BLK. TO BE TRANSMITTED
F406 F496 ISABX     FDB SABX  STORE "A" AT LOC. B+X
F408 F49E ILABX     FDB LABX  LOAD "A" WITH (LOC.B,X)
F40A F478 IABTX     FDB ABTX  ADD B TO X
F40C F463 IAITX     FDB AITX  ADD IMMEDIATE TO X
F40E F678 IRX2      FDB RX2   RECIEVE A CHAR. FROM ACIA
F410 F73D IPSHX     FDB PSX   PUSH X ONTO STACK
*
*
*      PROGRAM TO DISPLAY "ERROR"
*      - DISPLAYS "ERROR" FOR 2 SEC.
*      - THEN CLEARS THE DISPLAY RAM
*      - USES RTI AS A RETURN
*
F412 01          ERROR  NOP
F413 CE F423      LDX      #ERRORT - X=LOC. OF "ERROR"
F416 BD FFD6      JSR      DTSR2   - DISPLAY "ERROR"
F419 BD FFF1      JSR      DELAY2
F41C 03FF        FDB      $03FF   - DELAY 2 SEC.
F41E BD FFD0      JSR      CLDISP
F421 01          NOP
F422 3B          RTI
*  END  *
F423 7A          ERRORT FCB      $7A      E %01111010
F424 12          FCB      $12,$12    R,R %00010010
F425 12
F426 1E          FCB      $1E      O %00011110
F427 12          FCB      $12,0,0,0  R, 3 BLANKS
F428 00
F429 00
F42A 00
*  END  *
*
F42B          RM3RST EQU *  RESET FOR ROUTINES IN ROM3
*
F42B 86 03          LDA A  #$03
F42D B7 8408        STA A  ACI1CR MASTER RESET THE ACIA
F430 CE 0013        LDX      #BBUF1
F433 6F 00          R3RSL1 CLR      0,X      - CLEAR VARIABLES
F435 08            INX
F436 8C 004A        CPX      #TXBLKP+3

```



```

F439 26 F8          BNE      R3RSL1
*
F43B 7C 0015        INC      BUF1SR - INDICATE BUF1 EMPTY
F43E 7C 001E        INC      BUF2SR - INDICATE BUF2 EMPTY
F441 CE 1000        LDX      #BUF2
F444 DF 16          STX      BBUF2  - INIT. BUF2 POINTERS
F446 DF 1A          STX      EBUF2
F448 7E F3D0        JMP      RM4RST GO TO RESET FOR ROM4
**  END  **
*
*
F44B ACIH EQU * ASYNCHRONOUS COMMUNICATIONS INTERRUPT
*          HANDLER
*
* - HANDLES INTERRUPTS FROM THE ACIA, (MC6850)
* - IF 'IR' AND 'RDRE' ARE SET, USE RXPTR TO GO
*   TO A RX ROUTINE
* - IF 'IR' AND 'TDRE' ARE SET, USE TXPTR TO GO
*   TO A TX ROUTINE
* - RDRF MUST BE CHECKED FIRST AS RX INT. IS ALWAYS
*   ENABLED WHILE TX INT. MAY BE DISABLED EVEN
*   THOUGH TDRE MAY BE SET
* - ACIH ASSUMES DATA CARRIER DETECT INTERRUPTS
*   DO NO OCCUR
* - OVERFLOWS ARE HANDLED AS RECEIVES
* - FRAMING ERRORS ARE CLEARED AND IGNORED
* - ACI1SR IN "A" UPON ENTRY
*
F44B 85 01          BIT A    #$01      DO WE HAVE AN RX?
F44D 27 0C          BEQ      ACIHS1    BRANCH IF NOT
*
F44F 85 10          BIT A    #$10      FRAMING ERROR?
F451 26 04          BNE      ACIHX1    BRANCH IF SO
*
F453 DE 25          LDX      RXPTR     ASSUME READY TO RX
F455 6E 00          JMP      0,X      GO TO RX ROUTINE
*
F457 B6 8409 ACIHX1 LDA A    ACI1RX    CLEAR RX INTERRUPT
F45A 3B             RTI              EXIT
*
F45B 85 02          ACIHS1  BIT A    #$02      TEST IF WE CAN TX
F45D 27 F8          BEQ      ACIHX1    EXIT IF NOT
F45F DE 23          LDX      TXPTR     GO TO TX ROUTINE
F461 6E 00          JMP      0,X
*
**  END  **
*
*
F463      AITX      EQU      *          ADD IMMEDIATE TO X
*
* - ADDS CONSTANT DATA TO X(MAIN)
* - CHANGES ONLY X IN MAIN
*
F463 30            TSX              X PTS. TO TOP OF STACK

```



```

F464 E6 04      LDA B 4,X      B = XL(MAIN)
F466 A6 03      LDA A 3,X      A = XH(MAIN)
F468 EE 05      LDX          5,X      X = RETURN TO MAIN
F46A 09         DEX
F46B 09         DEX
F46C 09         DEX
F46D 09         DEX          X = LOC. OF DATA
F46E EB 01      ADD B 1,X      ADD L. BYTE TO X(MAIN)
F470 A9 00      ADC A 0,X      ADD H. BYTE TO X(MAIN)
F472 30         TSX          X PTS. TO STACK TOP
F473 A7 03      STA A 3,X      STORE SUM IN X(MAIN)
F475 E7 04      STA B 4,X
F477 3B         RTI

```

```

*
**  END  **

```

```

*

```

```

F478      ABTX      EQU      *          ADD B TO X

```

```

*

```

```

* - ADDS ACC. B(MAIN) TO ACC. X(MAIN)

```

```

* - CHANGES ONLY X IN MAIN

```

```

*

```

```

F478 30         TSX          X PTS. TO STACK TOP
F479 A6 01      LDA A 1,X      A(CURRENT)=B(MAIN)
F47B AB 04      ADD A 4,X      ADD X(LOW BYTE MAIN)
F47D 24 02      BCC          ABTXS1 CHECK FOR CARRY
F47F 6C 03      INC          3,X      PROP. CARRY
F481 A7 04      ABTXS1 STA A 4,X      REPLACE X(L.B. MAIN)
F483 3B         RTI

```

```

**  END  **

```

```

*

```

```

*

```

```

* SWISR1      - SOFTWARE INTERRUPT SUBROUTINE ONE

```

```

*

```

```

* - PLACES X(FROM MAIN)+B(FROM MAIN) INTO X(CURRENT)

```

```

* - X MUST POINT TO TOP OF RETURN VECTORS TO MAIN

```

```

* - CHANGES NOTHING IN MAIN

```

```

* - CHANGES X,B(CURRENT)

```

```

*

```

```

F484 36         SWISR1 PSH A      ***  STORE A
F485 A6 03      LDA A 3,X      A = XHIGH FROM MAIN
F487 E6 04      LDA B 4,X      B = XLOW FROM MAIN
F489 EB 01      ADD B 1,X      B=XLOW(MAIN)+B(MAIN)
F48B 89 00      ADC A #$00      PROPAGATE CARRY
F48D 37         PSH B          **
F48E 36         PSH A          *  STORE X+B FROM MAIN
F48F 30         TSX          X = TOP OF STACK
F490 EE 00      LDX          0,X      X = X+B FROM MAIN
F492 31         INS          *
F493 31         INS          **  REMOVE JUNK FROM STACK
F494 32         PUL A          ***  RESTORE A
F495 39         RTS

```

```

**  END  **

```

```

*

```

```

*

```



```

*      SABX      - STORE ACC. A  AT  B,X
*
*      - STORES  A(MAIN) AT LOCATION B+X(MAIN)
*      - CHANGES NO REGISTERS IN MAIN
*      - CHANGES CURRENT A,B,X
*      - CALLING SEQUENCE:  SWI
*                          FDB 0,ISABX
*
F496 30          SABX      TSX          X = TOP OF STACK
F497 A6 02          LDA  A    2,X      A(CURRENT) = A(MAIN)
F499 8D E9          BSR      SWISR1
F49B A7 00          STA  A    0,X      STORE  AT B+X(MAIN)
F49D 3B          RTI
**      END      **
*
*
*      LABX      - LOAD ACC. A  FROM  B,X
*
*      - LOADS A(MAIN) WITH CONTENTS OF LOCATION B+X(MAIN)
*      - CHANGES A IN MAIN
*      - CHANGES CURRENT A,B,X
*      - CALLING SEQUENCE:  SWI
*                          FDB 0,ILABX
*
F49E 30          LABX      TSX          X = TOP OF STACK
F49F 8D E3          BSR      SWISR1    X = B+X(FROM MAIN)
F4A1 A6 00          LDA  A    0,X      LOAD FROM B+X(MAIN)
F4A3 30          TSX          X = TOP OF STACK
F4A4 A7 02          STA  A    2,X      PLACE IN A(MAIN)
F4A6 3B          RTI
**      END      **
*
*
*      BITOAS    - BINARY TO ASCII  ROUTINE
*
*      - SUBROUTINE TO CONVERT THE BINARY CONTENTS OF  A
*      TO INTO TWO ASCII CHARACTERS IN ACC.  A  AND  B
*      - RESULT OF HIGH NIBBLE IN  A , LOW IN  B
*      - CHANGES ONLY A, B, CCR
*
F4A7 36          BITOAS  PSH  A      ***  STORE BYTE
F4A8 44          LSR  A
F4A9 44          LSR  A      SHIFT HIGH NIBBLE LOW
F4AA 44          LSR  A
F4AB 44          LSR  A
F4AC 81 09        CMP  A    #$09
F4AE 2F 04        BLE      BIASS1    BRANCH IF .LE. $09
*
F4B0 8B 37        ADD  A    #$37      CONVERT TO ASCII
F4B2 20 02        BRA      BIASS2    HIGH CHARACTER IN A
*
F4B4 8B 30        BIASS1  ADD  A    #$30  CONVERT TO ASCII
F4B6 33          BIASS2  PUL  B      ***  ORIGINAL BYTE
F4B7 C4 0F        AND  B    #$0F      BET LOWER NIBBLE

```



```

F4B9 C1 09          CMP B   #$09
F4BB 2F 03          BLE     BIASS3  BRANCH IF .LE. $09
*
F4BD CB 37          ADD B   #$37    LOW CHARACTER IN B
F4BF 39             RTS
*
F4C0 CB 30          BIASS3 ADD B   #$30    LOW CHARACTER IN B
F4C2 39             RTS
*      47USEC.
**      END      **
*
*
*      STX1  - SWI ROUTINE:    TRANSMIT 1
*
* - ENTERED VIA A SWI
* - TRANSMITS THE BYTE IN ACC. A THROUGH ACIA #1
* - IF THE BYTE CANNOT BE TRANSMITTED IMMEDIATELY,
*   IT IS STORED IN A FIFO BUFFER CALLED BUF1
* - THE CHAR. IN BUF1 WILL BE TXED BY ROUTINE TX1
*   WHENEVER AN INTERRUPT FROM ACIA IND. IT IS READY
*   TO TRANSMIT
* - THE STATUS OF BUF1 IS IND. BY BITS 0,1 OF BUF1SR
*   BIT 0 = 1 IFF BUF1 IS EMPTY, (MT)
*   BIT 1 = 1 IFF BUF1 IS FULL
* - DOES NOT CHANGE ANY REGISTERS IN MAIN
*
F4C3 30             STX1    TSX          X = TOP OF STACK
F4C4 A6 02          LDA A   2,X        A = A FROM MAIN
F4C6 D6 15          LDA B   BUF1SR    TEST BIT 0 IF BUF1 MT
F4C8 C5 01          BIT B   #$01
F4CA 27 1D          BEQ     STX1S1    BRANCH IF NOT MT
*
F4CC C6 A9          LDA B   #$A9      BUF1 MT
F4CE F7 8408        STA B   ACI1CR    SET ACI1CR FOR:
*                               ENABLED TX AND RX INT.
*                               1/16 CLOCK (9600 BAUD), EVEN PARITY,
*                               EVEN PARITY, 7 BITS, 1 STOP
F4D1 7F 0020        CLR     TXCMP    IND. TX INT. ENABLED
F4D4 CE F535        LDX     #TX1     INIT. PTRS. TO TX AND
F4D7 DF 23          STX     TXPTR    RX ROUTINES
F4D9 CE F509        LDX     #RX1
F4DC DF 25          STX     RXPTR
F4DE F6 8408        LDA B   ACI1SR    AFTER SLIGHT DELAY,
F4E1 C5 02          BIT B   #$02     IS TDRE HIGH?
F4E3 27 04          BEQ     STX1S1    BRANCH IF NOT READY
*
F4E5 B7 8409        STA A   ACI1TX    TX CONTENTS OF A
F4E8 3B             RTI
*
F4E9 D6 14          STX1S1 LDA B   EBUF1  STORE AT END OF BUF1
F4EB CE 0700        LDX     #BUF1     IE) AT EBUF1+BUF1
F4EE 3F             SWI
F4EF 0000          FDB     0, ISABX
F4F1 F406

```



```

F4F3 5C          INC B
F4F4 D7 14      STA B   EBUF1   INCREMENT EBUF1
F4F6 7F 0015    CLR     BUF1SR  BUF1 NOT EMPTY
F4F9 D1 13      CMP B   BBUF1   TEST IF EBUF1=BBUF1
F4FB 26 47      BNE     TX1S1   BRANCH IF NOT FULL
*
F4FD 86 02      LDA A   #$02    EBUF1=BBUF1 SO FULL
F4FF 97 15      STA A   BUF1SR  SET BIT 1 OF BUF1SR
F501 CE F7AD    LDX     #BUFTBL DISP. "BUFFER FULL"
F504 BD FFD6    JSR
F507 20 3B      BRA     TX1S1
**   END   **
*
*
F509      RX1      EQU      *           RECEIVE 1
*
*   - STORES IN RXD THE ASCII CHARACTERS
*   LISTED BELOW IN THIER PRIORITY GROUPS
*   - CHARACTERS IN HIGHER PRIORITY GROUPS CAN
*   OVER WRITE THOSE IN LOWER GROUPS
*   - CHARACTERS IN THE SAME GROUP DO NOT
*   OVER WRITE
*   HIGHEST ("DC1", " ", "?"), ("%",">"),("#") LOWEST
*
F509 B6 8409    LDA A   AC11RX  CLEAR RX INTERRUPT
F50C 81 11      CMP A   #$11    SKIP IF "DC1" RXED
F50E 27 1C      BEQ     RX1X2
F510 81 5D      CMP A   #$5D    SKIP IF " " RXED
F512 27 18      BEQ     RX1X2
F514 81 3F      CMP A   #$3F    SKIP IF "?" RXED
F516 27 14      BEQ     RX1X2
F518 D6 49      LDA B   RXD
F51A 27 04      BEQ     RX1S1   SKIP IF RXD CLEAR
F51C C1 23      CMP B   #$23    EXIT IF "#" NOT RXED
F51E 26 0E      BNE     RX1X1
*
F520 81 25      RX1S1  CMP A   #$25    SKIP IF "%" RXED
F522 27 08      BEQ     RX1X2
F524 81 23      CMP A   #$23    SKIP IF "#" RXED
F526 27 04      BEQ     RX1X2
F528 81 3E      CMP A   #$3E
F52A 26 02      BNE     RX1X1   EXIT IF ">" NOT RXED
*
F52C 97 49      RX1X2  STA A   RXD      SET RXD
*
F52E 3B         RX1X1  RTI
**   END   **
F52F 0006      RMB     6
*
*
*   TX1      TRANSMIT ONE
*
* - ENTERED VIA AN INTERRUPT GENERATED BY ACIA #1
* - WILL TRANSMIT A BYTE AT A TIME FROM BUF1 UNTIL

```


* TDRE IN ACIA #1 GOES HIGH OR UNTIL BUF1 IS EMPTY
 * - SEE STX1
 * - DOES NOT CHANGE ANY REGISTERS IN MAIN

```

*
F535 D6 15 TX1 LDA B BUF1SR BIT 0 IND. IF BUF1 MT
F537 C5 01 BIT B #$01
F539 27 09 BEQ TX1S1 BRANCH IF NOT EMPTY
*
F53B 86 C9 LDA A #$C9 BUF1 MT, SET RTS HIGH,
F53D B7 8408 STA A ACI1CR
F540 7C 0020 INC TXCMP IND. TX INT. DISABLED
F543 3B RTI
*
F544 F6 8408 TX1S1 LDA B ACI1SR TEST TDRE OF ACIA
F547 C5 02 BIT B #$02
F549 26 01 BNE TX1S2 BRANCH IF HIGH
F54B 3B RTI LOW, CANNOT TX
F54C D6 13 TX1S2 LDA B BBUF1 WE CAN TX A BYTE
F54E CE 0700 LDX #BUF1 FROM BBUF1+BUF1
F551 3F SWI
F552 0000 FDB 0, ILABX
F554 F408
F556 B7 8409 STA A ACI1TX TRANSMIT
F559 5C INC B INC. FRONT OF BUF1
F55A D7 13 STA B BBUF1
F55C 7F 0015 CLR BUF1SR BUF1 NOT FULL
F55F D1 14 CMP B EBUF1 TEST IF BUF1 EMPTY
F561 26 E1 BNE TX1S1 NOT MT, TRY NEXT BYTE
F563 86 01 LDA A #$01 BUF1 MT, SET BIT 0
F565 97 15 STA A BUF1SR
F567 3B RTI RTS HIGH AND TX INT. WILL

```

** END ** BE DISABLED ON NEXT INT. WHEN TDRE IS HIGH

* TX1TST - TRANSMIT 1 TEST PROGRAM

*

* - GENERATES STRINGS OF 40 OF THE SAME CHARACTER
 * PRECEDED BY A BLANK AND FOLLOWED BY RETURN
 * - GENERATES STRINGS OF THE CHAR. FROM "0" TO "F"
 * - NUMBER OF TIMES THIS IS TO BE REPEATED MUST BE
 * STORED IN LOC. IXRV AS A TWO BYTE NUMBER

*

```

F568 86 0F TX1TST LDA A #$0F STARTING CHARACTER
F56A 36 TX1L2 PSH A *** STORE NIBBLE
F56B 36 PSH A ** TWICE
F56C 86 20 LDA A #$20 ASCII FOR A SPACE
F56E 3F SWI TRANSMIT THIS BYTE
F56F 0000 FDB 0, ISTX1
F571 F402
F573 32 PUL A ** GET NIBBLE
F574 8B 30 ADD A #$30 MAKE AN ASCII CHAR.
F576 C6 30 LDA B #$30 48 ITERATIONS
F578 3F TX1L1 SWI TRANSMIT THE BYTE
F579 0000 FDB 0, ISTX1
F57B F402
F57D 5A DEC B

```



```

F57E 26 F8          BNE      TXTL1
*
F580 CE F788        LDX      #ENDLIN TRANSMIT END OF LINE
F583 BD F5A5        JSR      TX1CHS
F586 BD FFF1        JSR      DELAY2 WAIT .5 SEC.
F589 00FF          FDB      $00FF
F58B 32             PUL      A      *** RESTORE NIBBLE
F58C 4A             DEC      A
F58D 2A DB          BPL      TXTL2
*
F58F DE 0D          LDX      IXRV      GET X ITERATION VALUE
F591 09             DEX
F592 DF 0D          STX      IXRV
F594 26 D2          BNE      TX1TST DO AGAIN IF NOT IXRV
*
F596 CE F7A5        LDX      #ENDTBL DISPLAY END
F599 BD FFD6        JSR      DTSR2
F59C CE F780        LDX      #EOFTBL TX "$ENDOFFILE"
F59F BD F5A5        JSR      TX1CHS
F5A2 3E             TXTWAI WAI      STOP
F5A3 20 FD          BRA      TXTWAI WAIT FOR DATA ENTRY
** END **

```

*

*

* TX1CHS - TRANSMIT A CHARACTER STRING

*

* - TRANSMITS A CHARACTER STRING THROUGH

* ACIA #1

* - WHEN CALLED WE REQUIRE X = LOC. OF FIRST CHAR.
* TO BE TRANSMITTED

* - CHANGES REGISTERS A,X

* - HAS NO TIME DELAY TO ALLOW FOR TRANSMISSION,

* OTHER PROGRAMMING MUST MAKE SURE THE BUFFER

* DOES NOT OVERFLOW

*

```

F5A5 A6 00 TX1CHS LDA A 0,X CHARACTER TO BE TX.
F5A7 81 04      CMP A #$04
F5A9 27 08      BEQ      TXCSX IF CHAR. = 4, EXIT

```

*

```

F5AB 3F          SWI          TRANSMIT BYTE
F5AC 0000        FDB      0,ISTX1

```

```

F5AE F402

```

```

F5B0 08          INX          X = LOC. OF NEXT

```

```

F5B1 20 F2        BRA      TX1CHS

```

*

```

F5B3 39          TXCSX RTS

```

** END **

*

*

```

F5B4          STX2 EQU * SWI TRANSMIT TWO

```

*

* - A SWI ROUTINE TO TRANSFER THE ASCII CHARACTERS

* IN TXBLK TO BUF2 SO THEY CAN BE TRANSMITTED.

* - STORES CHAR. UPTO AND INCLUDING THE FIRST "CR"


```

* - ENABLES THE TX2 ROUTINE, UPDATES BUF2SR
* - WHEN BUF2 BECOMES FULL, "BUF. FULL" IS DISPLAYED
*   AND WE RTI WITH CARRY SET.
* *IF BUF2 WAS EMPTY,
* *THEN -BUF2SR IS CLEARED
*       -IF RXPTR .NE. RX2
*       -THEN .TXPTR AND RXPTR ARE SET TO TX2 AND RX2
*           .RX FLAGS ARE CLEARED
*           .TXCMP IS CLEARED AND TXINT IS ENABLED
*       -ELSE IF DC1FLG OR NAFLG IS SET
*           THEN TXCMP IS CLEARED AND TXINT IS ENABLED
*           FI
*       -FI

```

```

* *FI

```

```

* - CHANGES ONLY CCR IN MAIN, C SET IF BUF2 FULL,
*   C CLEARED OTHERWISE

```

```

F5B4 0E          CLI          ALLOW FOR INTERRUPTS
F5B5 96 1E      LDA A  BUF2SR
F5B7 81 02      CMP A  #$02   TEST IF BUF2 IS FULL
F5B9 27 2B      BEQ          STX2X1 IF IT IS, EXIT
*
F5BB DE 1A      LDX          EBUF2  INIT. END OF BUF2 PTR.
F5BD DF 1C      STX          EBUF2T
F5BF CE 0560    LDX          #TXBLK X PTS. TO TXBLK
F5C2 A6 00      STX2L1 LDA A  0,X   GET CHARACTERS TO TX
F5C4 08          INX
F5C5 DF 47      STX          TXBLKP
F5C7 DE 1C      LDX          EBUF2T STORE CHAR.IN BUF2
F5C9 A7 00      STA A  0,X
F5CB 08          INX
F5CC 8C 1800    CPX          #BUF2ST INC. END BUFFER PTR.
F5CF 26 03      BNE          STX2S2
F5D1 CE 1000    LDX          #BUF2
F5D4 DF 1C      STX2S2 STX          EBUF2T
F5D6 9C 16      CPX          BBUF2  CHECK IF BUF2 FULL
F5D8 26 1E      BNE          STX2S3 BRANCH IF NOT FULL
*
F5DA 81 0D      CMP A  #$0D   WAS "CR" JUST STORED
F5DC 26 08      BNE          STX2X1 SKIP IF NOT
F5DE DF 1A      STX          EBUF2  UPDATE EBUF2 (TX)
F5E0 86 02      LDA A  #$02   INDICATE BUF2 FULL
F5E2 97 1E      STA A  BUF2SR
F5E4 20 1C      BRA          STX2X2 EXIT
F5E6 86 02      STX2X1 LDA A  #$02
F5E8 97 1E      STA A  BUF2SR INDICATE BUF2 FULL
F5EA CE F7AD    LDX          #BUFTBL
F5ED BD FFD6    JSR          DTSR2  DISPLAY "BUF. FULL"
F5F0 30          TSX
F5F1 A6 00      LDA A  0,X   SET CARRY IN RETURN
F5F3 8A 01      ORA A  #$01   ( AS DATA NOT TXED )
F5F5 A7 00      STA A  0,X
F5F7 3B          RTI

```

```

*

```



```

F5F8 DE 47      STX2S3 LDX      TXBLKP
F5FA 81 0D      CMP A    #$0D
F5FC 26 C4      BNE      STX2L1  BRANCH CHAR.WASN'T CR
*
F5FE DE 1C      LDX      EBUF2T  UPDATE EBUF2 POINTER
F600 DF 1A      STX      EBUF2
F602 30          STX2X2 TSX
F603 A6 00      LDA A    0,X      CLEAR CARRY IN RETURN
F605 84 FE      AND A    #$FE    ( AS DATA WAS TXED )
F607 A7 00      STA A    0,X
F609 01          NOP
F60A 96 1E      LDA A    BUF2SR
F60C 81 01      CMP A    #$01    TEST IF EMPTY
F60E 26 29      BNE      STX2X    SKIP IF NOT
*
F610 7F 001E    CLR      BUF2SR  BUF2 IS NO LONGER MT
F613 CE F678    LDX      #RX2
F616 9C 25      CPX      RXPTR   CHECK IF RX2/TX2 ACT.
F618 26 0A      BNE      STX2S0  SKIP IF NOT
F61A 96 22      LDA A    DC1FLG  CHECK IF "DC1" RXED
F61C 26 13      BNE      STX2SA  SKIP IF TRUE
F61E 96 21      LDA A    NAFLG   CHECK IF IN NA MODE
F620 26 0F      BNE      STX2SA  SKIP IF TRUE
F622 20 15      BRA      STX2X    DO NOT ENABLE TXINT
*
F624 DF 25      STX2S0 STX      RXPTR  SET RX,TX POINTERS
F626 CE F63A    LDX      #TX2
F629 DF 23      STX      TXPTR
F62B B6 8408    LDA A    ACI1SR  CLEAR RX FLAGS
F62E B6 8409    LDA A    ACI1RX
F631 86 29      STX2SA LDA A    #$29  TX ONLY,7BITS,EV.PAR.
F633 B7 8408    STA A    ACI1CR  (1/16)CLOCK,1STOP BIT
F636 7F 0020    CLR      TXCMP   IGNORE ACKNOWLEDGE
F639 3B          STX2X RTI      UNTIL LINE TXED
*      3385USEC. FOR 47 CHAR
**      END      **
*
*
F63A      TX2      EQU *      TRANSMIT TWO
*
* - A HALF DUPLEX TRANSMIT ROUTINE TO TRANSMIT A
*   SAMPLE BLOCK,( MAX NO. OF ASCII CHARACTERS
*   DETERMINED BY THE SIZE OF TXBLK.)
* - ASSUMES DATA IN ASCII
* - USES "CR" TO INDICATE END OF BLOCK
* - WHEN BUF2 BECOMES EMPTY, BIT 0 IN BUF2SR IS SET,
*   TXINT IS DISABLED, AND TX2CMP IS SET
* - IF WE ARE NOT IN NO ACKNOWLEDGE MODE,
*   AND THE LAST CHARACTER IN A STRING WAS TXED,
*   TXINT IS DISABLED AND TX2CMP IS SET
*
F63A 96 1E      LDA A    BUF2SR
F63C 81 01      CMP A    #$01    IS BUF2 EMPTY?
F63E 27 29      BEQ      TX2X2    EXIT IF MT

```



```

*
F640 7F 001E      CLR      BUF2SR  BUF2 CANNOT BE FULL
F643 DE 16        LDX      BBUF2   GET NEXT CHAR. TO TX
F645 A6 00        LDA A    0,X
F647 B7 8409      STA A    ACI1TX  TX CHAR.
F64A 08          INX
F64B 8C 1800      CPX      #BUF2ST
F64E 26 03        BNE      TX2S1   INCREMENT POINTER
F650 CE 1000      LDX      #BUF2
F653 DF 16        STX      BBUF2   SAVE POINTER
F655 9C 1A        CPX      EBUF2   CHECK IF EMPTY
F657 27 09        BEQ      TX2X1   EXIT IF EMPTY
*
F659 D6 21        LDA B    NAFLG   ARE WE IN NA MODE?
F65B 26 1A        BNE      TX2X3   EXIT IF SO
*
F65D 81 0D        CMP A    #$0D    CHECK IF "CR" WAS TXED
F65F 27 08        BEQ      TX2X2   EXIT IF IT WAS
*
F661 3B          RTI
*
F662 86 01        TX2X1  LDA A    #$01
F664 97 1E        STA A    BUF2SR  INDICATE BUF2 EMPTY
F666 7F 0022      CLR      DC1FLG  IND. WAITING FOR "DC1"
F669 B6 8408      TX2X2  LDA A    ACI1SR  CLEAR RX INT. FLG.
F66C B6 8409      LDA A    ACI1RX  AND RX REGISTER
F66F 86 C9        LDA A    #$C9
F671 B7 8408      STA A    ACI1CR  DISABLE TX, ENA. RX
F674 7C 0020      INC      TXCMP   ENABLE RX ROUTINE
*
F677 3B          TX2X3  RTI
*      95USEC./TX AVE., 2.3MSEC. MAX. DURING .025 SEC.
**      END      **
*
*
F678      RX2      EQU *   RECEIVE TWO
*
* - THE HALF DUPLEX RECEIVE ROUT. ASSOCIATED WITH TX2
* - FOR %PTIP=ON ( IE. NAFLG CLEAR ) :
* - IF "DC1" IS RXED WHILE BUF2 NOT MT AND TXCMP SET,
*   THEN TX INT. IS ENABLED AND TXCMP IS CLEARED
* - IF "DC1" IS RXED WHILE BUF2 IS EMPTY, DC1FLG
*   IS SET SO STX2 CAN ENABLE TXINT
* - OTHERWISE JUST THE RX INTERRUPT IS CLEARED
*
F678 B6 8409      LDA A    ACI1RX  CLEARS INTERRUPT
*
F67B 81 11        CMP A    #$11    CHECK FOR "DC1"
F67D 26 1F        BNE      RX2X1   EXIT IF NOT
F67F 96 1E        LDA A    BUF2SR  CHECK IF BUF2 EMPTY
F681 81 01        CMP A    #$01
F683 27 11        BEQ      RX2S1   SKIP IF EMPTY
F685 96 20        LDA A    TXCMP   CHECK IF TX COMPLETE
F687 27 15        BEQ      RX2X1   EXIT IF IT IS NOT

```



```

*
F689 96 21      LDA A   NAFLG      CHECK IF IN NA MODE
F68B 26 D5      BNE      TX2X1     EXIT IF WE ARE
*
F68D 86 29      LDA A   #$29      ENA.TX,DISA.RX,7BITS,
F68F B7 8408    STA A   ACI1CR    1STOP,EV.PAR.,(1/16)
F692 7F 0020    CLR      TXCMP    INDICATE TXING
F695 3B         RTI              EXIT
*
F696 7C 0022 RX2S1 INC      DC1FLG
F699 86 49      LDA A   #$49      DISABLE TX AND RX
F69B B7 8408    STA A   ACI1CR
F69E 3B         RX2X1 RTI              EXIT
**  END  **
*
*
F69F      TX2TST EQU *   TRANSMIT TWO TEST
*
* - A MAINLINE ROUTINE TO TEST THE TX2 ROUTINES
* - USE MONITOR TO SET  I)  IXRV TO # OF ITERATIONS
*                      II) ACC. A TO DESIRED PART OF
*                      FIRST ITERATION
*                      III) ACC. B FOR ACKN. MODE,
*                      ( CLR. B FOR NO ACK. MODE )
*
F69F 36         PSH A           ***
F6A0 37         PSH B           **  STORE B,A
F6A1 7C 0021    INC      NAFLG   ENTER NA MODE
F6A4 CE F78C    LDX      #PTIPON
F6A7 BD F71E    JSR      TX2CHS  TX '%PTIP=ON'
F6AA D6 1E      TX2TL  LDA B   BUF2SR
F6AC C1 01      CMP B   #$01
F6AE 26 FA      BNE      TX2TL   WAIT UNTIL BUF2 MT
F6B0 7F 0021    CLR      NAFLG   GET OUT OF NA MODE
F6B3 33         PUL B           **  RESTORE B
F6B4 5D         TST B
F6B5 26 03      BNE      TX2TS1
F6B7 7C 0021    INC      NAFLG   SET NAFLG IF CLEAR
F6BA 32         TX2TS1 PUL A           *** RESTORE A
F6BB CE 0490    TX2TL0 LDX      #Y    GET LOC.OF BLOCK
F6BE 36         PSH A           *** STORE A
F6BF A7 00      TX2TL1 STA A   0,X    GENERATE TEST BLOCK
F6C1 08         INX
F6C2 8C 04A8    CPX      #Y+$18    24 SAMPLE BYTES
F6C5 26 F8      BNE      TX2TL1
*
F6C7 CE 0490    LDX      #Y
F6CA BD F3F4    JSR      MDTI     MOVE Y TO BSIL (16)
F6CD BD F3EB    JSR      CKSM    CHECK SUM,(2 BYTES)
F6D0 BD F3E5    JSR      TMHB    ADD 3 TIME BYTES
F6D3 96 42      LDA A   FLGC     ADD FLAG CHARACTER
F6D5 B7 0666    STA A   BSIL+22  (22 HEX. BYTES TOTAL)
F6D8 7F 0042    CLR      FLGC
F6DB CE 0651    LDX      #BSIL+1  MOVE TO OUTPUT LINE

```



```

F6DE BD F3EE      JSR      GTXB      GEN.TXBLK(48 BYTES)
F6E1 3F           SWI
F6E2 0000         FDB      0,ISTX2  TRANSMIT TXBLK
F6E4 F404
F6E6 86 90        LDA A    #$90      WRITE TO 0 WITH AUTO.
F6E8 B7 8001      STA A    PKDICR
F6EB 32           PUL A      ***      RESTORE A
F6EC 36           PSH A      ***      STORE A
F6ED BD FFC4      JSR      WRITE2    DISPLAY DATA TXED
F6F0 BD FFF1      JSR      DELAY2    WAIT .5 SEC.
F6F3 00FF         FDB      $00FF
F6F5 32           PUL A      ***      RESTORE A
F6F6 4A           DEC A
F6F7 26 C2        BNE      TX2TLO    LOOP IF A NOT ZERO
*
F6F9 DE 0D        LDX      IXRV
F6FB 09           DEX
F6FC DF 0D        STX      IXRV
F6FE 26 BB        BNE      TX2TLO    LOOP IF IXRV NOT 0
*
F700 96 1E        TX2TL2 LDA A    BUF2SR  WAIT UNTIL BUF2 MT
F702 81 01        CMP A    #$01
F704 26 FA        BNE      TX2TL2
*
F706 7C 0021      INC      NAFLG
F709 CE F780      LDX      #EOFTBL  TX "$ENDFILE"
F70C BD F71E      JSR      TX2CHS
F70F CE F798      LDX      #PTIPOF  TX %PTIP=OFF
F712 BD F71E      JSR      TX2CHS
F715 CE F7A5      LDX      #ENDTBL  DISPLAY "END"
F718 BD FFD6      JSR      DTSR2
F71B 3E           TX2WAI WAI          STOP
F71C 20 FD        BRA      TX2WAI    ALLOW FOR DATA ENTRY
**      END      **
*
*
F71E      TX2CHS EQU * TRANSMIT TWO CHARACTER STRING
*
*      -  ALLOWS FOR THE TRANSMISSIONS OF AN ARBITRARY
*      CHARACTER STRINGS COMPATABLE TO TX2
*      -  MAX. CHAR. NUMBER SET BY SIZE OF TXBLK
*      -  $0D IE) "CR" IS LAST CHARACTER IN LINE
*      -  (X) ON ENTRY IS LOC. OF CHAR. STRING
*      -  CHANGES A,CCR,X
*
F71E DF 27        STX      TX2CSX
F720 CE 0560      LDX      #TXBLK
F723 DF 47        STX      TXBLKP
*
F725 DE 27        TX2CSL LDX      TX2CSX
F727 A6 00        LDA A    0,X      GET CHARACTER
F729 08           INX
F72A DF 27        STX      TX2CSX
F72C DE 47        LDX      TXBLKP

```



```

F72E A7 00      STA A 0,X      STORE CHAR. IN TXBLK
F730 08         INX
F731 DF 47      STX      TXBLKP
F733 81 0D      CMP A  #$0D      WAS CHARACTER "CR"?
F735 26 EE      BNE      TX2CSL   LOOP IF NOT

```

*

```

F737 3F         SWI
F738 0000       FDB      0,ISTX2  TRANSMIT BLOCK
F73A F404
F73C 39         RTS

```

** END **

*

*

```

F73D      PSHX   EQU * PUSH X

```

*

* - A SWI DRIVEN ROUTINE TO PUSH X ONTO THE STACK
 * - CHANGES NOTHING (EXCEPT SP) IN CALLING CONTEXT

*

```

F73D 34         DES      MAKE PLACE FOR X ON STACK
F73E 34         DES      CURRENT X PTS. TO RETURN
F73F 30         TSX      CONTEXT

```

*

```

F740 A6 02      LDA A 2,X      SHIFT RETURN CONTEXT
F742 A7 00      STA A 0,X      UP BY 2 BYTES
F744 A6 03      LDA A 3,X
F746 A7 01      STA A 1,X
F748 A6 04      LDA A 4,X
F74A A7 02      STA A 2,X
F74C A6 05      LDA A 5,X
F74E A7 03      STA A 3,X
F750 A6 06      LDA A 6,X
F752 A7 04      STA A 4,X
F754 A6 07      LDA A 7,X
F756 A7 05      STA A 5,X
F758 A6 08      LDA A 8,X
F75A A7 06      STA A 6,X

```

*

```

F75C A6 03      LDA A 3,X      PLACE X(MAIN) ON STACK
F75E E6 04      LDA B 4,X
F760 A7 07      STA A 7,X
F762 E7 08      STA B 8,X
F764 3B         RTI          EXIT

```

*

* 121 USEC.

** END **

*

*

```

F765      CLBUF2 EQU * CLEAR BUF2

```

*

* - A SUBROUTINE TO EMPTY BUF2, THE TRANSMISSION
 * BUFFER USED BY TX2
 * - CHANGES BUF2SR,BBUF2,EBUF2,A,B,X

*

```

F765 07         TPA          SAVE INT. MASK

```



```

F766 0F          SEI          DISABLE STX2,TX2
F767 CE 1000     LDX          #BUF2
F76A DF 1A       STX          EBUF2      RESET BUF2 POINTERS
F76C DF 16       STX          BBUF2
F76E C6 01       LDA B        #$01      INDICATE BUF2 EMPTY
F770 D7 1E       STA B        BUF2SR
F772 06          TAP          RESTORE INT. MASK
F773 39          RTS

```

```

*
```

```

**  END  **

```

```

*
```

```

*
```

```

*
```

```

TABLES

```

```

F780          ORG          $F780

```

```

*
```

```

END OF FILE STRING.

```

```

*
```

```

F780 5B          EOFTBL FCB          $5B,$0D,$0A,$5B,$0D,$0A,0,0

```

```

F781 0D

```

```

F782 0A

```

```

F783 5B

```

```

F784 0D

```

```

F785 0A

```

```

F786 00

```

```

F787 00

```

```

*
```

```

*  END OF LINE STRING

```

```

*
```

```

F788          ENDLIN EQU          *

```

```

F788 0D          FCB          $0D,$0A,$7F,$04 - "CR",
F789 0A          "LF","DEL",END OF STRING

```

```

F78A 7F

```

```

F78B 04

```

```

F78C 25 PTIPON FCC          '%PTIP=ON'

```

```

F78D 50

```

```

F78E 54

```

```

F78F 49

```

```

F790 50

```

```

F791 3D

```

```

F792 4F

```

```

F793 4E

```

```

F794 0D          FCB $0D,$0A,$7F,$04 - "CR",
F795 0A          "LF","DEL",END OF STRING

```

```

F796 7F

```

```

F797 04

```

```

F798 25 PTIPOF FCC          '%PTIP=OFF'

```

```

F799 50

```

```

F79A 54

```

```

F79B 49

```

```

F79C 50

```

```

F79D 3D

```

```

F79E 4F

```

```

F79F 46

```

```

F7A0 46

```

```

F7A1 0D          FCB $0D,$0A,$7F,$04 - "CR",

```



```

F7A2 0A                                "LF", "DEL", END OF STRING
F7A3 7F
F7A4 04
*
*      TABLE TO DISPLAY  'END'
*
F7A5 7A      ENDTBL FCB      %01111010      E
F7A6 16      FCB      %00010110      N
F7A7 1F      FCB      %00011111      D
F7A8 00      FCB      0,0,0,0,0      BLANKS
F7A9 00
F7AA 00
F7AB 00
F7AC 00
*
*      TABLE TO DISPLAY  'BUFFER FULL'
*
F7AD 00      BUFTBL FCB      0,0      2 BLANKS
F7AE 00
F7AF BE      FCB      %10111110      B.
F7B0 00      FCB      0      BLANK
F7B1 72      FCB      %01110010      F
F7B2 3D      FCB      %00111101      U
F7B3 38      FCB      %00111000      L
F7B4 38      FCB      %00111000      L
*
*
F7E0      ORG      $F7E0
*****
*
*
*      JUMP      TABLE
*
F7E0 7E F4A7 VBITAS JMP      BITOAS
F7E3 7E F5A5 VTXCHS JMP      TX1CHS
F7E6 7E F412 VERROR JMP      ERROR
F7E9 7E F42B VRMRST JMP      RM3RST
F7EC 7E F44B VACIH  JMP      ACIH
F7EF 00      FCB      0,0,0
F7F0 00
F7F1 00
F7F2 7E F765      JMP      CLBUF2
F7F5 7E F71E VTX2CS JMP      TX2CHS
F7F8 7E F568      JMP      TX1TST
F7FB 7E F69F      JMP      TX2TST
      END

```


ROM4

M68SAM is the property of Motorola Spd, Inc.
Copyright 1974 by Motorola Inc.

Motorola M6800 Cross Assembler, Release 1.1

```

NAM      ROM4      SAMPLING AND MULTIPLICATION ROUTINES
*
*  SOURCE FILE LABELS
*
*
*  PIA LABELS
*
9000      DRE      EQU      $9000
9001      CRE      EQU      $9001
9002      DRF      EQU      $9002
9003      CRF      EQU      $9003      > PIA #3
*
*
*  TIMER ADDRESSES ( MC6840 )
*
7C06      TL3H     EQU      $7C06 - TIMER # 1 LATCHES
7C04      TL2H     EQU      $7C04
7C02      TL1H     EQU      $7C02
7C00      CR3      EQU      $7C00 - # 1 CONTROL REG.;WRITE ONLY.
7C01      CR2      EQU      $7C01
7C00      CR1      EQU      $7C00
7C01      TSTATR   EQU      $7C01 - # 1 STATUS REG.;READ ONLY.
9406      TL6H     EQU      $9406 - TIMER # 2 LATCHES
9404      TL5H     EQU      $9404
9402      TL4H     EQU      $9402
9400      CR6      EQU      $9400 - # 2 CONTROL REG.;WRITE ONLY.
9401      CR5      EQU      $9401
9400      CR4      EQU      $9400
9401      TSTR2    EQU      $9401 - # 2 STATUS REG.;READ ONLY.
*
*
*  PROGR. KEYBD. DISPL. INTERFACE ADDR. ( I8279-5 )
*
8001      PKDICR   EQU      $8001 - CONTROL REGISTER;WRITE ONLY
8001      PKDISR   EQU      $8001 - STATUS REGISTER;READ ONLY
8000      FIFO     EQU      $8000 - KEYBOARD INPUT RAM;READ ONLY
8000      DISRAM   EQU      $8000 - DISPLAY RAM; READ/WRITE
*
*
*  MULTIPLIER ADDRESSES
*
A400      MCR      EQU      $A400 - MULTIPLIER CONTROL REGISTER
*  MCR : I
*  BIT 7(TCY /TCX / RS /RND /TRIL/TRIM/CLKY/CLKX)BIT 0
*  I
*  TRIL,TRIM ACTIVE LOW
A000      LPDRH    EQU      $A000 - HIGH BYTE -

```



```

A001    LPDRL    EQU $A001 - LOW  BYTE      > LOWER MULT.
A002    LPCRH    EQU $A002 - HIGH CONT. REG.> PIA  #4
A003    LPCRL    EQU $A003 - LOW CONT. REG. -
A004    UPDRH    EQU $A004 - HIGH BYTE      -
A005    UPDRL    EQU $A005 - LOW  BYTE      > UPPER MULT.
A006    UPCRH    EQU $A006 - HIGH CONT. REG.> PIA  #5
A007    UPCRL    EQU $A007 - LOW CONT. REG. -
A800    MIDRH    EQU $A800 - HIGH BYTE      -
A801    MIDRL    EQU $A801 - LOW  BYTE      > MULT. INPUTS
A802    MICRH    EQU $A802 - HIGH CONT. REG.> PIA  #6
A803    MICRL    EQU $A803 - LOW CONT. REG. -

```

*

*

*

MONITOR GLOBAL VARIABLES AND FLAGS

*

```

0004    ENTRY1   EQU $0004 - ENTRIES VIA ENTR FUNCTION
0005    ENTRY2   EQU $0005
0006    ENTRY3   EQU $0006
0007    ENTRY4   EQU $0007
0008    ENTRY5   EQU $0008
0009    ENTRY6   EQU $0009
000A    ENTRY7   EQU $000A
000B    ENTRY8   EQU $000B
000D    IXRV     EQU $000D,E -X REG. STORAGE, MAIN LINES
0010    MONFLG   EQU $0010 - IND. MAINLINE STOPED BY MONITOR
0011    SECFLG   EQU $0011 - IND. 2ND KEYBD. FNT. SEL.

```

*

*

*

SUBROUTINE LABLES

*

```

FFCA    DSPSN    EQU      $FFCA
FFCD    DSPDP    EQU      $FFCD
FFD0    CLDISP   EQU      $FFD0
FFD3    DTSR1    EQU      $FFD3
FFD6    DTSR2    EQU      $FFD6
FFD9    DSBISR   EQU      $FFD9
FFDC    BIDSSR   EQU      $FFDC
FFDF    DSPXAB   EQU      $FFDF
FFE2    DSTOBI   EQU      $FFE2
FFE5    BITODS   EQU      $FFE5
F7E0    BITOAS   EQU      $F7E0
FFC1    READ2    EQU      $FFC1
FFC4    WRITE2   EQU      $FFC4
FFE8    READ4    EQU      $FFE8
FFEB    WRITE4   EQU      $FFEB
FFEE    CNPIA1   EQU      $FFEE
FFF1    DELAY2   EQU      $FFF1
FF60    DATTBL   EQU      $FF60
FBF6    SCNDSR   EQU      $FBF6
FFC7    RDWL     EQU      $FFC7
EFD0    FILT     EQU      $EFD0
FFBE    FLSH     EQU      $FFBE

```

*

*

* SPECIAL JUMP TABLE ENTRIES

*

FBF0	TRAP	EQU	\$FBF0	
FBF3	STOP	EQU	\$FBF3	
FFF4	RM1RST	EQU	\$FFF4	- RESET TO ROM1
E000	RM6RST	EQU	\$E000	
BFD0	SPP1	EQU	\$BFD0	
BFD3	SPLP2	EQU	\$BFD3	
E721	CALP	EQU	\$E721	

*

*

*

* LINKAGE TABLE ENTRIES FOR SWI ROUTINES

*

F400	IERR	EQU	\$F400
F402	ISTX1	EQU	\$F402
F404	ISTX2	EQU	\$F404
F406	ISABX	EQU	\$F406
F408	ILABX	EQU	\$F408
F40A	IABTX	EQU	\$F40A
F40C	IAITX	EQU	\$F40C

*

*

* VARIABLES USED ONLY BY ROM 4

*

0032		ORG	\$0032	
0032	0002	SPLVEC	RMB 2	MAIN
0034	0001	ROFF	RMB 1	INT
0035	0002	IXRVI1	RMB 2	INT
0037	0002	IXRVM	RMB 2	TWO BYTE STORGE FOR MULT.
0039	0002	MDTIX	RMB 2	X STORAGE FOR MDTI
003B	0002	GTXBX	RMB 2	X STORAGE USED BY GTXB

*

*

* OTHER VARIABLES USED BY ROM 4

*

0040		ORG	\$0040
0040	0002	BSILP	RMB 2
0042	0001	FLGC	RMB 1
0043	0002		RMB 2
0045	0002		RMB 2
0047	0002	TXBLKP	RMB 2

*

*

* CONSTANTS USED BY ROM3

*

0400	S	EQU	\$0400	- LOCATION OF SAMPLE BLOCK
0490	Y	EQU	\$0490	- FILTER OUTPUT
0560	TXBLK	EQU	\$0560	- TX BLOCK LOCATION
0650	BSIL	EQU	\$0650	- BMS INPUT LINE
0690	BSOL	EQU	\$0690	- BMS OUTPUT LINE

*

*


```

F000                                ORG      $F000
*****
*
*
F000      RM4RST EQU      *          ROM 4 RESET
*
*   - CONFIGUERS PIA ON A/D BOARD
*   - SETS UP PROGRAMMABLE TIMERS #1 AND #2
*
F000 7F 9001      CLR      CRE      CONFIGURE PIA PORTS E,F
F003 7F 9003      CLR      CRF      GET ACCESS TO DDR'S
F006 7F 9002      CLR      DRF      DRF IS AN INPUT
F009 86 07        LDA A    #$07
F00B B7 9000      STA A    DRE      3 LSB'S OF DRE ARE OUTPUTS
F00E 86 2C        LDA A    #$2C    ENABLE DRF,DRE,1USEC PULSE
F010 B7 9001      STA A    CRE      FROM CE2 BY READ ON DRE
F013 B7 9003      STA A    CRF      FROM CF2 BY WRITE TO DRF
*
*      PROGRAMMABLE TIMER # 2  SETUP
*      - Q6 FOR HOURS
*      - Q5 FOR MINUTES.
*      - Q4 FOR SECONDS
*      IRQ (DIS/EN)ABLED 0/1-  -----ALL CONTINUOUS MODE
*      ALL OUTS. ENABLED---- II I  -----(N/ML) MODE
*                               III II -----(PHI2/EXT.CLK.)
*                               III III          BIT 0:
*      CR6 = $90   =          %10010000----- NO DIV. BY 8
*      CR5 = $95   =          %10010101-----WRITE(CR3/CR1)
*      CR4 = $94   =          %10010100----- (OP./HALT)TIMER
*
*      T6=24 HR.  =24Q5   (N+1)=24    N= TL6H = $0017
*      T5=60 MIN.=60Q4   (M+1)(L+1)=(6)(10)  M,L=TL5H=$0509
*      T4=60 SEC.=60Q1   (M+1)(L+1)=(6)(10)  M,L=TL4H=$0509
*
F016 CE 0017 PTMSET LDX      #$0017
F019 FF 9406        STX      TL6H
F01C CE 0509        LDX      #$0509
F01F FF 9404        STX      TL5H
F022 FF 9402        STX      TL4H
F025 86 94          LDA A    #$94
F027 B7 9400        STA A    CR6
F027 B7 9400        STA A    CR6
F02A 86 95          LDA A    #$95
F02C B7 9401        STA A    CR5
F02F 86 94          LDA A    #$94
F031 B7 9400        STA A    CR4
*
*
*      PROGRAMMABLE TIMER # 1  SETUP
*      - Q1 = 1 HZ, Q2 = 50 HZ, Q3 = 200 HZ
*      - Q1 FOR TIME COUNTERS
*      - Q2 FOR SAMPLE FREQ.
*      - Q3 FOR 200 BAUD RATE TO HP PLOTTER
*      IRQ (DIS/EN)ABLED 0/1-  -----ALL CONTINUOUS MODE

```



```

* ALL OUTPUTS ENABLED----- II I -----(N/ML) MODE
*                               III II -----(PHI2/EXT.CLK.)
*                               III III          BIT 0:
*      CR3 = $92   =      %10010010-----NO DIV. BY 8
*      CR2 = $D7   =      %11010111-----WRITE(CR3/CR1)
*      CR1 = $90   =      %10010000----- (OPERATE/HALT)
*
* T3=.005SEC. = 5000 ; (N+1)=2500 ; N = TL3H = $09C3
* T2=.0250SEC.= 25,000;(M+1)(L+1)=(250)(100);
*                               M,L=TL2H=$F963
* T1=1 SEC      = 200T3 ; (N+1)=100 ; N = TL1H = $0063
*
F034 CE 09C3      LDX      #$09C3
F037 FF 7C06      STX      TL3H
F03A CE F963      LDX      #$F963      SET CLK.2 TO 40 HZ.
F03D FF 7C04      STX      TL2H
F040 CE 0063      LDX      #$0063
F043 FF 7C02      STX      TL1H
F046 86 92        LDA A    #$92
F048 B7 7C00      STA A    CR3
F04B 86 97        LDA A    #$97      DISABLE CLK.2 INT.
F04D B7 7C01      STA A    CR2
F050 86 90        LDA A    #$90
F052 B7 7C00      STA A    CR1
*
F055 C6 CC        LDA B    #$CC      MULT. CIRCUIT RESET
F057 F7 A400      STA B    MCR      DIS. MULT. TRIST. BUF.
F05A CE 0000      LDX      #$0000    CONFIG. PIA'S #4,#5,#6
F05D FF A802      STX      MICRH
F060 FF A006      STX      UPCRH
F063 FF A002      STX      LPCRH
F066 FF A004      STX      UPDRH      #5 IS AN INPUT
F069 FF A000      STX      LPDRH      #4 IS AN INPUT
F06C CE FFFF      LDX      #$FFFF
F06F FF A800      STX      MIDRH      #6 IS AN OUTPUT
F072 CE 0404      LDX      #$0404
F075 FF A006      STX      UPCRH
F078 CE 0606      LDX      #$0606    CONFIGURE INT. INPUTS
F07B FF A802      STX      MICRH
F07E CE 3404      LDX      #$3404
F081 FF A002      STX      LPCRH      ENABLE MULT. INPUTS
*
F084 7E E000      JMP      RM6RST    GO TO ROM 6 RESET
**      END      **
*
F087      PTMH EQU *      PROGRAMMABLE TIMER HANDLER
*
* - CHECKS FOR PROPER TIMER INTERRUPT
* - COMPARES SPLVEC WITH SPLTE1 TO SEE IF WE ARE
*   IN THE SAMPLE TEST MODE.
* - IF MONFLG IS SET, TIMER INT. IS CLEARED AND WE RTI
* - OTHERWISE CONT. PASSES TO SAMPLE PROCESS DRIVER
*
F087 B6 7C01      LDA A    TSTATR    GET PTM #1 STATUS

```



```

F08A 85 02          BIT A  #02    TEST FOR CLK.2 INT.
F08C 26 0A          BNE      PTMHS1  BRANCH IF WE DO
*
F08E FE 7C06        LDX      TL3H
F091 FE 7C04        LDX      TL2H      CLEAR ALL POSSIBLE
F094 FE 7C02        LDX      TL1H      TIMER INTERRUPTS
F097 3B            PTMHX1 RTI
*
F098 FE 7C04 PTMHS1 LDX      TL2H      CLEAR INT. FROM CLK.2
F09B DE 32          LDX      SPLVEC
F09D 8C BFD3        CPX      #SPLP2    SKIP IF SAMPLE
FOA0 26 03          BNE      PTMHS2    PROC. 2 NOT ACTIVE
FOA2 7E BFD3        JMP      SPLP2     GOTO SAMPLE PROC. 2
FOA5 8C E721 PTMHS2 CPX      #CALP     SKIP IF CAL.
FOA8 26 03          BNE      PTMHS3    INACTIVE
FOAA 7E E721        JMP      CALP      GO TO CAL. PROC.
FOAD 8C F1ED PTMHS3 CPX      #SPLTE1   CHECK IF IN TEST MD.
FOB0 26 07          BNE      SPLPDF    BRANCH IF NOT
FOB2 96 10          LDA      A  MONFLG
FOB4 26 E1          BNE      PTMHX1     EXIT IF M.L. STOPED
*
FOB6 7E F1ED        JMP      SPLTE1    JUMP TO SAMPLE TEST
*
FOB9 7E BFD0 SPLPDF JMP      SPP1      DEFLT. TO SAMP. PROC.1
*
**  END  **
FOBC 000C          RMB      12
*
*
FOC8      CKSM      EQU      *          CHECK SUM
*
*  - SUBROUTINE TO CALCULATE THE CHECK SUM FOR 8
*  TWO BYTE SAMPLES IN BSIL AND STORE IT IN
*  BSIL(17,18)
*  - USES 16 BIT ADDITION
*  - CHANGES A,B,CCR,X
*  - ASSUMES THE FIRST BYTE OF BSIL IS RESERVED
*
FOC8 CE 0651        LDX      #BSIL+1  3 GET LOC. OF BLOCK
FOCB 4F            CLR      A          2
FOCC 5F            CLR      B          2/7
*
FOCD EB 01 CKSML1   ADD      B  1,X    5 GENERATE CHECK SUM
FOCF A9 00          ADC      A  0,X    5
F0D1 08            INX          4
F0D2 08            INX          4
F0D3 8C 0661        CPX      #BSIL+17 3 ADD FIRST 16 BYTES
F0D6 26 F5          BNE      CKSML1    4/25*8=200
*
F0D8 A7 00          STA      A  0,X    6 SUM IN LOC. 17,18
F0DA E7 01          STA      B  1,X    6
F0DC 39            RTS          5/17
*  224 USEC.
**  END  **

```



```

*
*
F0DD GTXB EQU * GENERATE TX BLOCK
*
* - CONVERTS THE 22 HEX BYTES AT LOCATION (X)
* INTO 44 ASCII BYTES IN TXBLK
* - ADDS 6 ASCII TIME BYTES TO THE ABOVE
* - ADDS 4 OPTIONAL CHARACTER FLAGS TO THE ABOVE
* - ADDS A CARRIGE RETURN, "CR", TO THE ABOVE
* - ASSUMES THE FIRST BYTE OF BSIL IS RESERVED
* - CHANGES A,B,X
* - INPUTS : (X) LOCATION OF 22 HEX DATA BYTES
* - OUTPUTS: TXBLK LOCATION OF 48 BYTE ASCII BLOCK
*
F0DD DF 3B STX GTXBX SAVE LOC. OF DATA
F0DF 86 20 LDA A #$20 FIRST CHAR. A BLANK
F0E1 B7 0560 STA A TXBLK
F0E4 CE 0561 LDX #TXBLK+1
F0E7 DF 47 STX TXBLKP TXBLK POINTER
F0E9 DE 3B GTXBL2 LDX GTXBX GET LOC. OF DATA
F0EB A6 00 LDA A 0,X CONV. BLOCK TO ASCII
F0ED 08 INX
F0EE DF 3B STX GTXBX
F0F0 BD F7E0 JSR BITOAS CONV. MS BYTE TO ASCII
F0F3 DE 47 LDX TXBLKP
F0F5 A7 00 STA A 0,X
F0F7 08 INX
F0F8 E7 00 STA B 0,X
F0FA 08 INX
F0FB DF 47 STX TXBLKP
F0FD 8C 058D CPX #TXBLK+45 GENERATE 45 CHAR.
F100 26 E7 BNE GTXBL2
*
F102 86 0D LDA A #$0D GEN. CARRIGE RETURN
F104 A7 00 STA A 0,X ( 46 CHAR. TOTAL )
F106 39 RTS
*
** END **
*
*
F107 MDTI EQU * MOVE DATA TO INPUT
*
* - TAKES THE 3 BYTE WORD FILTER OUTPUTS IN Y
* AND ROUNDS THEM UP TO 2 BYTES AND MOVES THEM
* TO BSIL, THE BUBBLE SYSTEM INPUT LINE
* - ASSUMES THE FIRST BYTE OF BSIL IS RESERVED
* - INPUTS : (X) - LOCATION OF 3 BYTE DATA
* - OUTPUTS: BSIL - LOCATION OF 2 BYTE DATA
* - CHANGES A,B,CCR,X
*
F107 DF 39 STX MDTIX 5 SAVE LOC. OF DATA
F109 CE 0651 LDX #BSIL+1 3 SET POINTER TO
F10C DF 40 STX BSILP 5/13 BSIL LOCATION
*

```



```

F10E DE 39      MDTIL1 LDX      MDTIX 4 GET LOC. OF FILT OUT
F110 A6 00      LDA A 0,X      5
F112 08         INX           4
F113 E6 00      LDA B 0,X      5 GET LSB
F115 08         INX           4
F116 6D 00      TST      0,X    7 ROUNDOFF ?
F118 2A 04      BPL      MDTIS1 4
F11A CB 01      ADD B #$01     1 ROUND UP
F11C 89 00      ADC A  #$00     1
F11E 08         MDTIS1 INX      4
F11F DF 39      STX      MDTIX 5 Y BLOCK POINTER
F121 DE 40      LDX      BSILP 4
F123 A7 00      STA A 0,X      6
F125 08         INX           4
F126 E7 00      STA B 0,X      6
F128 08         INX           4
F129 DF 40      STX      BSILP 5 DO UNTIL 16
F12B 8C 0661    CPX      #BSIL+17 3 BYTES PRODUCED
F12E 26 DE      BNE      MDTIL1 4/80*8=640
F130 39         RTS           5/5

```

* .658 MSEC.

** END **

*

*

F131 TMHB EQU * TIMER HEX BYTES

*

* - THIS ROUTINE STORES THE CURRENT TIME, HH.MM.SS
 * AS HEX BYTES AND STORES IT IN LOCATIONS
 * BSIL+19 TO BSIL+21
 * - CHANGES CCR,A,B,X

*

```

F131 36      PSH A*** MAKE SPACE FOR HH.MM.SS
F132 36      PSH A**
F133 36      PSH A*
F134 BD F144 JSR      TMSR      GET HH.MM.SS
F137 32      PUL A* LOAD SS
F138 B7 0665 STA A BSIL+21 STORE SS
F13B 32      PUL A** LOAD MM
F13C B7 0664 STA A BSIL+20 STORE MM
F13F 32      PUL A*** LOAD HH
F140 B7 0663 STA A BSIL+19 STORE HH
F143 39      RTS

```

* 159USEC.

** END **

*

*

F144 TMSR EQU * TIMER SUBROUTINE

*

* - THIS ROUTINE TAKES THE 3 COUNTER VALUES IN
 * PTM # 2 WHICH REPRESENT HRS. MIN. & SEC.
 * AND PRODUCES 3 HEX BYTES
 * - THESE THREE BYTES ARE PLACED ON THE STACK SO
 * THE CALLING ROUTINE CAN PULL THEM OFF
 * IN THE FOLLOWING ORDER: SS,MM,HH


```

*      - CHANGES CCR, A, B, X
*
F144 30          TSX      X PTS. TO RETURN VEC. ON STACK
F145 4F          CLR A
F146 F6 9406     LDA B    TL6H      READ COUNTER 6
F149 C6 17       LDA B    #$17     IGNORE TOP BYTE
F14B F0 9407     SUB B    TL6H+1    HOURS IN B AS HEX
*
F14E C5 10       BIT B    #$10     CONVERT B TO BCD
F150 27 02       BEQ      TMHBS1
F152 86 16       LDA A    #$16     ADD 16 TO A
F154 C4 0F       TMHBS1 AND B    #$0F  MASK OFF HIGH NIBBLE
F156 C1 0A       TMHBL1 CMP B    #$0A  B MUST BE .LT. $0A
F158 2D 06       BLT      TMHBS2    SKIP IF TRUE
F15A 5A         DEC B
F15B 8B 01       ADD A    #$01
F15D 19         DAA
F15E 20 F6       BRA      TMHBL1
*
F160 1B         TMHBS2 ABA          SUM BCD IN A
F161 19         DAA
F162 A7 04       STA A    4,X      STORE HH
*
F164 86 05       LDA A    #$05
F166 C6 09       LDA B    #$09
F168 B0 9404     SUB A    TL5H     GET UPPER DIGIT OF MIN.
F16B F0 9405     SUB B    TL5H+1   GET LOWER DIG. OF MIN.
F16E 48         ASL A
F16F 48         ASL A      PLACE UPPER DIG. IN
F170 48         ASL A      UPPER NIBBLE
F171 48         ASL A
F172 1B         ABA          FORM MINUTES IN A
F173 A7 03       STA A    3,X      STORE MM
*
F175 86 05       LDA A    #$05
F177 C6 09       LDA B    #$09
F179 B0 9402     SUB A    TL4H     GET UPPER DIGIT OF SEC.
F17C F0 9403     SUB B    TL4H+1   GET LOWER DIG. OF SEC.
F17F 48         ASL A
F180 48         ASL A      PLACE UPPER DIGIT IN
F181 48         ASL A      UPPER NIBBLE
F182 48         ASL A
F183 1B         ABA          FORM SECONDS IN A
F184 A7 02       STA A    2,X      STORE SS
F186 39         RTS
* 105USEC
*
**  END  **
*
*
F187      SAMP    EQU      *      SAMPLE
*
* - SUBROUTINE TO PLACE A BLOCK OF 8 24 BIT SAMPLES
*   AT LOCATION S ( ONLY 12 BITS SIGNIFICANT )

```



```

*      - +2.5V=$1FFC00 , 0.0V=$000000 , -2.5V=$E00000
*
F187 CE 0400      LDX      #S
F18A 5F          CLR B          START WITH INPUT #1
*
F18B 37          SAMPL1 PSH B*** STORE INPUT ID.
F18C F7 9000      STA B DRE      SELECT ANALOG INPUT
F18F B6 9000      LDA A DRE      ACTIVATE SAMLE/HOLD
F192 01          NOP          2 DELAY
F193 F7 9002      STA B DRF      5 ACT. HOLD,A/D 6USEC.
F196 26 08        BNE          SAMPS1 4 BR. IF NOT 1ST SAMP.
*
F198 AC 00        CPX      0,X
F19A AC 00        CPX      0,X      18 DELAY FOR 26 USEC.
F19C AC 00        CPX      0,X
F19E 20 14        BRA      SAMPS2 4
*
F1A0 E6 00        SAMPS1 LDA B 0,X 5 GET PREV. SAMPLE LSB
F1A2 09          DEX          4
F1A3 A6 00        LDA A 0,X 5      MSB
F1A5 43          COM A          2
F1A6 53          COM B          2  COPMLEMENT SAMPLE
F1A7 47          ASR A          2
F1A8 56          ROR B          2  SH.LOW AND PROP. SIGN
F1A9 C4 FC        AND B #$FC 2  CLEAR LOW 2 BITS
F1AB A7 00        STA A 0,X 6
F1AD 08          INX          4  STORE SAMPLE
F1AE E7 00        STA B 0,X 6
F1B0 08          INX          4
F1B1 6F 00        CLR      0,X 7  CLEAR LOW BYTE
F1B3 08          INX          4
*
F1B4 B6 9002      SAMPS2 LDA A DRF  GET HIGH BITS OF SAMPLE
F1B7 F6 9000      LDA B DRE  GET LOW BITS,ACT. SAMPLE
F1BA A7 00        STA A 0,X
F1BC 08          INX          STORE BITS
F1BD E7 00        STA B 0,X
F1BF 33          PUL B***
F1C0 5C          INC B
F1C1 C1 08        CMP B #$08 DO UNTIL 8 SAMPLES READ
F1C3 26 C6        BNE          SAMPL1
*
F1C5 E6 00        LDA B 0,X 5  CONVERT LAST SAMPLE
F1C7 09          DEX          4
F1C8 43          COM A          2
F1C9 53          COM B          2  COPMLEMENT SAMPLE
F1CA 47          ASR A          2
F1CB 56          ROR B          2  SHIFT LOW,PROP.SIGN
F1CC C4 FC        AND B #$FC 2  CLEAR LOW 2 BITS
F1CE A7 00        STA A 0,X 6
F1D0 08          INX          4  STORE SAMPLE
F1D1 E7 00        STA B 0,X 6
F1D3 6F 01        CLR      1,X 7  CLEAR THIRD BYTE
F1D5 39          RTS

```



```

* 899USEC.
** END **
*
*
F1D6 SPLT EQU * SAMPLE TEST
*
* - PROGRAM TO TEST SAMPLE ROUTINE
* - CONFIGURES CLK.2 TO INTERRUPT AT SAMPLING FREQ.
* - SETS SPLVEC TO SPLTE1
* - USES THE SAMP ROUTINE TO CREATE A +-2.5V DVM
* - INPUT CHANNEL SELECTED BY USING MONITOR TO STORE
* THE CHANNEL NO. IN ACC. A
*
F1D6 CE F1ED LDX #SPLTE1 STORE SAMP. TEST LOC.
F1D9 DF 32 STX SPLVEC IN SPLVEC
F1DB 86 C7 LDA A #$C7 ENABLE CLK.2 AND INT.
F1DD B7 7C01 STA A CR2
F1E0 BD FFD0 JSR CLDISP CLEAR DISPLAY
F1E3 86 01 LDA A #$01 USE CH.#1 AS DEFAULT
*
F1E5 3E SPLTWA WAI M.L. LOOP FOR INTS.
F1E6 20 FD BRA SPLTWA
F1E8 7F 7C01 CLR CR2 MONITOR USED TO STOP
F1EB 20 F8 BRA SPLTWA DEPRESS STOP, SCAN, SCAN,
* STRT
*
F1ED SPLTE1 EQU * INTERRUPT SAMPLE TEST ROUTINE
F1ED BD F187 JSR SAMP OBTAIN SAMPLE BLOCK
F1F0 30 TSX
F1F1 E6 02 LDA B 2,X GET CHAN. # FROM
F1F3 5A DEC B ACC. A(MAIN)
F1F4 17 TBA
F1F5 58 ASL B OBTAIN DISPLACEMENT OF
F1F6 1B ABA SAMPLE (MULT. BY 3)
F1F7 16 TAB
F1F8 CE 0400 LDX #S
F1FB 3F SWI
F1FC 0000 FDB 0,IABTX ADD DISPLACEMENT
F1FE F40A SAMPLE BLOCK LOC.
F200 A6 00 LDA A 0,X
F202 E6 01 LDA B 1,X GET SAMPLE MSB,LSB
F204 7F 0034 CLR ROFF
*
* SAMPLE IS TO BE MULT. BY 2.50122 = %10.1000000001
* AFTER DIVIDING BY 4
F207 47 ASR A
F208 56 ROR B BIV. BY 2.0
F209 97 35 STA A IXRVI1
F20B D7 36 STA B IXRVI1+1 STORE IN 2 BYTE LOC.
F20D 47 ASR A
F20E 56 ROR B GET ORIGINAL SAMPLE
F20F 47 ASR A
F210 56 ROR B MULT. BY 0.5
F211 79 0034 ROL ROFF STORE LOW BIT IN ROFF

```



```

F214 DB 36      ADD B IXRVI1+1
F216 99 35      ADC A IXRVI1 FORM 2.5 TIMES SAMPLE
F218 97 35      STA A IXRVI1
F21A D7 36      STA B IXRVI1+1
F21C E6 00      LDA B 0,X ORIG. HIGH SAMP. BYTE
F21E 57          ASR B EFFECTIVELY MULT. BY 2EXP(-10)
F21F 24 03      BCC SPLTS1
F221 7C 0034     INC ROFF MAINTAIN ROUNDOFF BITS
F224 57          SPLTS1 ASR B
F225 79 0034     ROL ROFF
F228 5D          TST B PROP. SIGN OF B THROUGH A
F229 2B 03      BMI SPLTS2
F22B 4F          CLR A
F22C 20 02      BRA SPLTS3
F22E 86 FF      SPLTS2 LDA A #$FF
F230 DB 36      SPLTS3 ADD B IXRVI1+1 GENERATE 2.50122
F232 99 35      ADC A IXRVI1 TIMES SAMPLE
F234 58          ASL B DIVIDE BY 2EXP(11)
F235 49          ROL A BY PLACING M.S. NIB.IN
F236 7C 0034     INC ROFF TOP NIBBLE OF ACC. A
F239 77 0034     ASR ROFF
F23C DB 34      ADD B ROFF ADD ROUNDOFF BITS
F23E 89 00      ADC A #$00 FINAL HEX RESULT IN A,B

```

*

```

F240 BD F25A     JSR H2DC CONV. A,B HEX TO BCD
F243 36          PSH A*** STORE BCD MSB
F244 37          PSH B**  STORE BCD LSB
F245 07          TPA
F246 36          PSH A*   STORE CCR
F247 BD FFD0     JSR CLDISP CLEAR DISPLAY
F24A 32          PUL A*   RESTORE CCR
F24B 06          TAP
F24C BD FFCA     JSR DSPSN DISPLAY SIGN
F24F 33          PUL B**  RESTORE BCD SAMPLE
F250 32          PUL A***
F251 BD FFEB     JSR WRITE4 DISPLAY SAMPLE
F254 C6 01      LDA B #$01
F256 BD FFCD     JSR DSPDP PLACE DEC. PT. IN
F259 3B          RTI CHAR. #1

```

** END **

*

*

F25A H2DC EQU * HEX TWO TO DECIMAL

*

* - CHANGES TWO BYTES OF TWO'S COMPLEMENT INTO

* TWO BYTES OF UNSIGNED BCD ASSUMING NIBBLE

* 16EXP(0) IS THE TOP NIBBLE IN ACC. A

* - THE N BIT OF CCR WILL BE SET IF HEX WAS NEGATIVE

* - THE V BIT OF CCR WILL BE SET IF AN OVERFLOW

* OCCURED DURING THE CONVERSION

* - HIGH BYTES IN A, LOW BYTES IN B

* - CHANGES A,B,X,CCR

*

F25A 36 PSH A*1* 5,X STORE SIGN OF HEX NUM.


```

F25B 4D          TST A
F25C 2A 07       BPL      H2DCS1 BRANCH IF SIGN PLUS
*
F25E 5A          DEC B
F25F 82 00       SBC A   #$00   CNG. TO POSITIVE
F261 53          COM B           UNSIGNED HEX
F262 43          COM A
F263 2B 78       BMI H2DCX1   IF STILL NEG., WE HAVE
*                   OVERFLOW, EXIT (*1*)
F265 37          H2DCS1 PSH B*2* 4,X   STORE UNSIGNED LSB
F266 36          PSH A*3* 3,X           MSB
F267 C6 50       LDA B   #$50   FOR ROUND OFF
F269 37          PSH B*4* 2,X           LB (0.000XX)
F26A 5F          CLR B   CLEAR SPACE ON STACK FOR BCD
F26B 37          PSH B*5* 1,X           MB (0.0XX )
F26C 37          PSH B*6* 0,X           HB (X.X )
*
F26D 30          TSX
F26E E6 04       LDA B   4,X           GET LSB OF SAMPLE
F270 C4 0F       AND B   #$0F         GET LOW NIBBLE
F272 27 10       BEQ      H2DCS2     SKIP IF ZERO
*
F274 A6 02       H2DCL1 LDA A   2,X   GET LB
F276 8B 24       ADD A   #$24   CONV. LOW NIB. OF LSB
F278 19          DAA
F279 A7 02       STA A   2,X
F27B A6 01       LDA A   1,X
F27D 89 00       ADC A   #$00   MAX. VALUE = 0.00410
F27F A7 01       STA A   1,X
F281 5A          DEC B
F282 26 F0       BNE      H2DCL1 DO UNTIL LOW NIB. 0
*
F284 E6 04       H2DCS2 LDA B   4,X   GET LSB
F286 54          LSR B
F287 54          LSR B
F288 54          LSR B           GET HIGH NIB. IN LOW
F289 54          LSR B           NIB. POSITION
F28A 27 11       BEQ      H2DCS3
*
F28C A6 02       H2DCL2 LDA A   2,X   CONV. HIGH NIB. OF LSB
F28E 8B 91       ADD A   #$91
F290 19          DAA
F291 A7 02       STA A   2,X
F293 A6 01       LDA A   1,X
F295 89 03       ADC A   #$03
F297 19          DAA           MAX. VALUE = 0.06275
F298 A7 01       STA A   1,X
F29A 5A          DEC B
F29B 26 EF       BNE      H2DCL2
*
F29D E6 03       H2DCS3 LDA B   3,X   GET MSB
F29F C4 0F       AND B   #$0F         GET LOW NIBBLE
F2A1 27 18       BEQ      H2DCS4
*

```



```

F2A3 A6 02      H2DCL3 LDA A  2,X
F2A5 8B 50      ADD A  #$50
F2A7 19         DAA
F2A8 A7 02      STA A  2,X
F2AA A6 01      LDA A  1,X
F2AC 89 62      ADC A  #$62
F2AE 19         DAA
F2AF A7 01      STA A  1,X
F2B1 A6 00      LDA A  0,X
F2B3 89 00      ADC A  #$00
F2B5 19         DAA
F2B6 A7 00      STA A  0,X
F2B8 5A         DEC B
F2B9 26 E8      BNE      H2DCL3
*
F2BB A6 03      H2DCS4 LDA A  3,X      GET MSB
F2BD 84 F0      AND A  #$F0      GET HIGH NIBBLE
F2BF AB 00      ADD A  0,X
F2C1 19         DAA
F2C2 A7 00      STA A  0,X
F2C4 07         TPA
F2C5 6D 05      TST      5,X
F2C7 2B 04      BMI      H2DCS5      BRANCH IF MINUS
F2C9 84 F7      AND A  #$F7      CLEAR N BIT
F2CB 20 02      BRA      H2DCS6
F2CD 8A 08      H2DCS5 ORA A  #$08      SET N BIT
F2CF 06         H2DCS6 TAP
F2D0 24 03      BCC      H2DCS7      RESTORE CCR
F2D2 0B         SEV
F2D3 20 01      BRA      H2DCS8      SET V IF C SET
F2D5 0A         H2DCS7 CLV
F2D6 32         H2DCS8 PUL A*6*      CLEAR V IF C CLEAR
F2D7 33         PUL B*5*      BCD HB IN A
F2D8 31         INS *4*      BCD LB IN B
F2D9 31         INS *3*
F2DA 31         INS *2*
F2DB 31         INS *1*      ADJUST STACK
F2DC 39         RTS
*
F2DD 32         H2DCX1 PUL A*1*      RESTORE SIGN
F2DE 4D         TST A      CLEAR OR SET N BIT
F2DF 0B         SEV      SET OVERFLOW BIT
F2E0 39         RTS
**  END  **
*
*
F2E1 MF21 EQU * MULTIPLY FRACTIONAL 2'S COMPLEMENT 1
*
* - THIS ROUTINE MULTIPLIES 2 16 BIT FIXED POINT
*   NUMBERS TO PRODUCE A 16 BIT FIXED POINT PRODUCT
* - ASSUMES THE MULTIPLIED NUMBERS, X AND Y, ARE IN
*   FRACTIONAL 2'S COMPLEMENT NOTATION AND PRODUCES A
*   ROUNDED OFF FRACTIONAL 2'S COMPLEMENT PRODUCT, P.
* - IT ASSUMES X IS A 24 BIT NUMBER AND

```



```

*   ROUNDS IT TO 16 BITS
* - IT THEN ADDS THE PRODUCT TO THE CONTENTS OF THE
*   LOCATION WHERE THE PRODUCT IS TO BE STORED
* - THE LOCATION OF X, Y, AND THE LOCATION WHERE P IS
*   TO BE STORED ARE DEFINED IN THE SUBROUTINE CALL
*   AS FOLLOWS :
*       JSR MF21
*       FDB X,Y,P
*       ( NEXT INSTRUCTION )
* - ROUTINE USES THE TRW/MPY-16HJ MULTIPLIER CHIP
*   AND(*DOES NOT ALLOW FOR INTERRUPTS AS IT*)IS NOT
*   REENTRANT AND IN NOT CONCURRENT WITH MF22
* - CHANGES A,B,X,CCR (*DOES NOT CHANGE CCR*)
*
*( TPA          2   STORE CCR )*
*( SEI          2/4  DISABLE INTERRUPTS )*
*
F2E1 30          TSX          4   X PTS. TO RET. VEC.
F2E2 EE 00       LDX          0,X 6   X IS RETURN LOC.
F2E4 DF 37       STX          IXRVM 5   STORE RETURN LOC.
*
F2E6 EE 00       LDX          0,X 6   X IS MULTIPLICAND LOC.
F2E8 E6 02       LDA B       2,X 5   CHECK FOR OVERFLOW
F2EA 2A 05       BPL          MF21S14
*
F2EC EE 00       LDX          0,X 6
F2EE 08          INX          4
F2EF 20 02       BRA          MF21S2 4
*
F2F1 EE 00       MF21S1 LDX          0,X 6   X IS MULTIPLICAND
F2F3 FF A800     MF21S2 STX          MIDRH 6   INPUT X
F2F6 C6 CD       LDA B       #$CD 2
F2F8 F7 A400     STA B       MCR      5/53 CLOCK IN X
*
F2FB DE 37       LDX          IXRVM 4   X IS THE RET. LOC.
F2FD EE 02       LDX          2,X 6   X IS MULTILPIER LOC.
F2FF EE 00       LDX          0,X 6   X IS MULTIPLIER
F301 FF A800     STX          MIDRH 6   INPUT MULTIPLIER, Y
F304 C6 CE       LDA B       #$CE 2   CLOCK IN Y
F306 F7 A400     STA B       MCR      5
F309 C6 3C       LDA B       #$3C 2   DISABLE MULT. INPUT
F30B F7 A002     STA B       LPCRH 5
F30E C6 C0       LDA B       #$C0 2   ENABLE BOTH MULT. O/P
F310 F7 A400     STA B       MCR      5/43
*
F313 DE 37       LDX          IXRVM 4   X IS RET. LOC.
F315 EE 04       LDX          4,X 6   X IS LOC. OF PROD., P
F317 F6 A001     LDA B       LPDRL 4   GET LOWEST BIT
F31A 58          ASL B          2
F31B F6 A000     LDA B       LPDRH 4
F31E 59          ROL B          2   REMOVE RED. SIGN BIT,
F31F EB 02       ADD B       2,X 5   ADD LOW BIT
F321 E7 02       STA B       2,X 6   ADD HIGH PROD. TO
F323 E6 01       LDA B       1,X 5   CONTENTS OF PROD.LOC.

```



```

F325 F9 A005      ADC B   UPDRL 4 ADD LOW BYTE OF HIGH
F328 E7 01      STA B   1,X   6 MULT. OUTPUT
F32A E6 00      LDA B   0,X   5
F32C F9 A004      ADC B   UPDRH 4 ADD HIGH BYTE
F32F E7 00      STA B   0,X   6/ 63
*
F331 C6 CC      LDA B   #$CC  2 DISABLE MULT. O/PS
F333 F7 A400      STA B   MCR   5
F336 C6 34      LDA B   #$34  2 ENABLE MULT. INPUT
F338 F7 A002      STA B   LPCRH 5
F33B 31         INS           4 ADJUST STACK
F33C 31         INS           4 AS RTS NOT USED
F33D DE 37      LDX      IXRVM 4 X IS RET. LOC.
* (TAP          2  RESTORE CCR ) *
F33F 6E 06      JMP      6,X   4/32 RETURN
*
*      TIME                      189USEC.
*      80(189USEC.) = 15.12MSEC.
**  END  **
*
*

```

```

F341 MF22 EQU * MULTIPLY FRACTIONAL 2'S COMPLEMENT 2

```

```

*
* - THIS ROUTINE MULTIPLIES 2 16 BIT FIXED POINT
*   NUMBERS TO PRODUCE A 16 BIT FIXED POINT PRODUCT
* - ASSUMES THE MULTIPLIED NUMBERS, X AND Y, ARE IN
*   FRACTIONAL 2'S COMPLEMENT NOTATION AND PRODUCES A
*   ROUNDED OFF FRACTIONAL 2'S COMPLEMENT PRODUCT, P.
* * IT ALSO ASSUMES X HAS ALREADY BEEN LATCHED INTO
*   THE MULTIPLIER CHIP
* - IT THEN ADDS THE PRODUCT TO THE CONTENTS OF THE
*   LOCATION WHERE THE PRODUCT IS TO BE STROED
* - THE LOCATION OF Y AND THE LOCATION WHERE P IS
*   TO BE STORED ARE DEFINED IN THE SUBROUTINE CALL
*   AS FOLLOWS :
*       JSR MF21
*       FDB Y,P
*       ( NEXT INSTRUCTION )
* - THIS ROUTINE USES THE TRW/ MPY-16HJ MULTIPLIER
*   CHIP AND (*DOES NOT ALLOW FOR INTERRUPTS AS IT*)
*   IS NOT REENTRANT NOR CONCURRENT WITH MF21
* - CHANGES A,B,X,CCR (*DOES NOT CHANGE CCR*)
*

```

```

* ( TPA          2  STORE CCR ) *
* ( SEI          2/4  DISABLE INTERRUPTS ) *
*

```

```

F341 30         TSX           4 X PTS.TO RETURN VEC.
F342 EE 00      LDX      0,X   6 X IS RETURN LOC.
F344 DF 37      STX      IXRVM 5 STORE RETURN LOC.
*
F346 EE 00      LDX      0,X   6 X IS MULT. LOC.
F348 EE 00      LDX      0,X   6 X IS MULTIPLIER
F34A FF A800     STX      MIDRH 6 INPUT MULTIPLIER,Y
F34D C6 CE      LDA B   #$CE  2 CLOCK IN Y

```



```

F34F F7 A400      STA B   MCR      5
F352 C6 3C        LDA B   #$3C    2 DISABLE MULT. INPUT
F354 F7 A002      STA B   LPCRH   5
F357 C6 C0        LDA B   #$C0    2 ENABLE BOTH MULT.O/P
F359 F7 A400      STA B   MCR      5/54
*
F35C DE 37        LDX      IXRVM   4 X IS RET. LOC.
F35E EE 02        LDX      2,X     6 X IS LOC. OF PROD.,P
F360 F6 A001      LDA B   LPDRL   4 CHECK FOR ROUNDOFF
F363 58           ASL B           2 GET LOWEST BIT
F364 F6 A000      LDA B   LPDRH   4
F367 59           ROL B           2 REMOVE RED.SIGN BIT,
F368 EB 02        ADD B   2,X     5 APP. LOWEST BIT
F36A E7 02        STA B   2,X     6 ADD TO CONTENTS
F36C E6 01        LDA B   1,X     5 OF PROD. LOC.
F36E F9 A005      ADC B   UPDRL   4 ADD LOW BYT
F371 E7 01        STA B   1,X     6 OF UPPER MULT O/P
F373 E6 00        LDA B   0,X     5
F375 F9 A004      ADC B   UPDRH   4 ADD HIGH BYTE
F378 E7 00        STA B   0,X     6/ 63
*
F37A C6 CC        LDA B   #$CC    2 DISABLE MULT. O/P
F37C F7 A400      STA B   MCR      5
F37F C6 34        LDA B   #$34    2 ENABLE MULT. INPUT
F381 F7 A002      STA B   LPCRH   5
F384 31           INS           4 ADJUST STACK
F385 31           INS           4 AS RTS NOT USED
F386 DE 37        LDX      IXRVM   4 X IS RET. LOC.
* ( TAP          2  RESTORE CCR ) *
F388 6E 04        JMP      4,X     4/32 RETURN
*
*      TIME                      147USEC.
*      80(147USEC.) = 11.76MSEC.
**    END    **
*
*
F3D0              ORG      $F3D0
*****
*
*      JUMP  TABLE
*
F3D0 7E F000      JMP      RM4RST
F3D3 7E F087      JMP      PTMH
F3D6 7E F187      JMP      SAMP
F3D9 7E F25A      JMP      H2DC
F3DC 7E F1D6      JMP      SPLT
F3DF 7E F2E1      JMP      MF21
F3E2 00           FCB      0,0,0
F3E3 00
F3E4 00
F3E5 7E F131      JMP      TMHB
F3E8 7E F341      JMP      MF22
F3EB 7E F0C8      JMP      CKSM
F3EE 7E F0DD      JMP      GTXB

```


F3F1 7E F144
F3F4 7E F107

JMP TMSR
JMP MDTI
END

ROM5

M68SAM is the property of Motorola Spd, Inc.
Copyright 1974 by Motorola Inc.

Motorola M6800 Cross Assembler, Release 1.1

NAM ROM5 DIGITAL FILTER ROUTINE
* AND BUBBLE MEMORY BOOTLOOP WRITE ROUTINE
*
*
*
* SUBROUTINE LABLES
*

FFCA	DSPSN	EQU	\$FFCA
FFCD	DSPDP	EQU	\$FFCD
FFD0	CLDISP	EQU	\$FFD0
FFD3	DTSR1	EQU	\$FFD3
FFD6	DTSR2	EQU	\$FFD6
FFD9	DSBISR	EQU	\$FFD9
FFDC	BIDSSR	EQU	\$FFDC
FFDF	DSPXAB	EQU	\$FFDF
FFE2	DSTOBI	EQU	\$FFE2
FFE5	BITODS	EQU	\$FFE5
F7E0	BITOAS	EQU	\$F7E0
FFC1	READ2	EQU	\$FFC1
FFC4	WRITE2	EQU	\$FFC4
FFE8	READ4	EQU	\$FFE8
FFEB	WRITE4	EQU	\$FFEB
FFEE	CNPIA1	EQU	\$FFEE
FFF1	DELAY2	EQU	\$FFF1
FF60	DATTBL	EQU	\$FF60
FBF6	SCNDSR	EQU	\$FBF6
FFC7	RDWL	EQU	\$FFC7
FFBE	FLSH	EQU	\$FFBE
F3F1	TMSR	EQU	\$F3F1
F3DF	MF21	EQU	\$F3DF
F3E8	MF22	EQU	\$F3E8
F7E3	TX1CHS	EQU	\$F7E3
F7F5	TX2CHS	EQU	\$F7F5

*
*
* LABLES FOR KEYBOARD ROUTINES
*
*

F7E6	ERROR	EQU	\$F7E6
------	-------	-----	--------

*
*
* LINKAGE TABLE ENTRIES FOR SWI ROUTINES
*

F400	IERR	EQU	\$F400
F402	ISTX1	EQU	\$F402
F404	ISTX2	EQU	\$F404
F406	ISABX	EQU	\$F406
F408	ILABX	EQU	\$F408
F40A	IABTX	EQU	\$F40A

F40C IAITX EQU \$F40C

*
*
*
*
*
*

VARIABLES USED ONLY BY ROM5

INTERMEDIATE FILTER DATA ARRAYS :

0418	W	EQU	\$0418
0430	O	EQU	\$0430
0448	P	EQU	\$0448
0460	Q	EQU	\$0460
0478	R	EQU	\$0478

*
*
*
*
*
*

OTHER VARIABLES USED BY ROM 5

0490	Y	EQU	\$0490
0400	S	EQU	\$0400

*
*
*
*

COEFFICIENT ARRAYS :

04B8	A0	EQU	\$04B8
04C8	A1	EQU	\$04C8
04D8	A2	EQU	\$04D8
04E8	B1	EQU	\$04E8
04F8	B2	EQU	\$04F8
0508	C0	EQU	\$0508
0518	C1	EQU	\$0518
0528	C2	EQU	\$0528
0538	D1	EQU	\$0538
0548	D2	EQU	\$0548

*
*
*
*

BUBBLE MEMORY SYSTEM LOCATIONS

AC00	BCDR	EQU	\$AC00	BUB. CONTROLER DATA REG.
AC01	BCSR	EQU	\$AC01	BUB. CONT. STATUS REG.

*
*
*

*

E800 ORG \$E800

*
*

E800 FILT EQU * FILTER

*
*
*
*

- 4TH ORDER DIGITAL FILTER ROUTINE IMPLEMENTED
USING 2 2ND ORDER 2D FILTERS
(SEE IEEE MICRO FEB. 1981 P. 410)

* - SEE PSEUDOCODE AND FILTER SPECS.
 * - CHANGES CCR, A, B, X
 * - THIS ROUTINE IS NON-REENTRANT
 *

E800	FE	0430	LDX	O+0	W = O
E803	FF	0418	STX	W+0	
E806	F6	0432	LDA B	O+0+2	
E809	F7	041A	STA B	W+0+2	
E80C	FE	0448	LDX	P+0	O = P
E80F	FF	0430	STX	O+0	
E812	F6	044A	LDA B	P+0+2	
E815	F7	0432	STA B	O+0+2	
E818	FE	0460	LDX	Q+0	Y = Q
E81B	FF	0490	STX	Y+0	
E81E	F6	0462	LDA B	Q+0+2	
E821	F7	0492	STA B	Y+0+2	
E824	FE	0478	LDX	R+0	Q = R
E827	FF	0460	STX	Q+0	
E82A	F6	047A	LDA B	R+0+2	
E82D	F7	0462	STA B	Q+0+2	
E830	FE	0433	LDX	O+3	W = O
E833	FF	041B	STX	W+3	
E836	F6	0435	LDA B	O+3+2	
E839	F7	041D	STA B	W+3+2	
E83C	FE	044B	LDX	P+3	O = P
E83F	FF	0433	STX	O+3	
E842	F6	044D	LDA B	P+3+2	
E845	F7	0435	STA B	O+3+2	
E848	FE	0463	LDX	Q+3	Y = Q
E84B	FF	0493	STX	Y+3	
E84E	F6	0465	LDA B	Q+3+2	
E851	F7	0495	STA B	Y+3+2	
E854	FE	047B	LDX	R+3	Q = R
E857	FF	0463	STX	Q+3	
E85A	F6	047D	LDA B	R+3+2	
E85D	F7	0465	STA B	Q+3+2	
E860	FE	0436	LDX	O+6	W = O
E863	FF	041E	STX	W+6	
E866	F6	0438	LDA B	O+6+2	
E869	F7	0420	STA B	W+6+2	
E86C	FE	044E	LDX	P+6	O = P
E86F	FF	0436	STX	O+6	
E872	F6	0450	LDA B	P+6+2	
E875	F7	0438	STA B	O+6+2	
E878	FE	0466	LDX	Q+6	Y = Q
E87B	FF	0496	STX	Y+6	
E87E	F6	0468	LDA B	Q+6+2	
E881	F7	0498	STA B	Y+6+2	
E884	FE	047E	LDX	R+6	Q = R
E887	FF	0466	STX	Q+6	
E88A	F6	0480	LDA B	R+6+2	
E88D	F7	0468	STA B	Q+6+2	
E890	FE	0439	LDX	O+9	W = O
E893	FF	0421	STX	W+9	

E896	F6	043B	LDA	B	O+9+2	
E899	F7	0423	STA	B	W+9+2	
E89C	FE	0451	LDX		P+9	O = P
E89F	FF	0439	STX		O+9	
E8A2	F6	0453	LDA	B	P+9+2	
E8A5	F7	043B	STA	B	O+9+2	
E8A8	FE	0469	LDX		Q+9	Y = Q
E8AB	FF	0499	STX		Y+9	
E8AE	F6	046B	LDA	B	Q+9+2	
E8B1	F7	049B	STA	B	Y+9+2	
E8B4	FE	0481	LDX		R+9	Q = R
E8B7	FF	0469	STX		Q+9	
E8BA	F6	0483	LDA	B	R+9+2	
E8BD	F7	046B	STA	B	Q+9+2	
E8C0	FE	043C	LDX		O+12	W = O
E8C3	FF	0424	STX		W+12	
E8C6	F6	043E	LDA	B	O+12+2	
E8C9	F7	0426	STA	B	W+12+2	
E8CC	FE	0454	LDX		P+12	O = P
E8CF	FF	043C	STX		O+12	
E8D2	F6	0456	LDA	B	P+12+2	
E8D5	F7	043E	STA	B	O+12+2	
E8D8	FE	046C	LDX		Q+12	Y = Q
E8DB	FF	049C	STX		Y+12	
E8DE	F6	046E	LDA	B	Q+12+2	
E8E1	F7	049E	STA	B	Y+12+2	
E8E4	FE	0484	LDX		R+12	Q = R
E8E7	FF	046C	STX		Q+12	
E8EA	F6	0486	LDA	B	R+12+2	
E8ED	F7	046E	STA	B	Q+12+2	
E8F0	FE	043F	LDX		O+15	W = O
E8F3	FF	0427	STX		W+15	
E8F6	F6	0441	LDA	B	O+15+2	
E8F9	F7	0429	STA	B	W+15+2	
E8FC	FE	0457	LDX		P+15	O = P
E8FF	FF	043F	STX		O+15	
E902	F6	0459	LDA	B	P+15+2	
E905	F7	0441	STA	B	O+15+2	
E908	FE	046F	LDX		Q+15	Y = Q
E90B	FF	049F	STX		Y+15	
E90E	F6	0471	LDA	B	Q+15+2	
E911	F7	04A1	STA	B	Y+15+2	
E914	FE	0487	LDX		R+15	Q = R
E917	FF	046F	STX		Q+15	
E91A	F6	0489	LDA	B	R+15+2	
E91D	F7	0471	STA	B	Q+15+2	
E920	FE	0442	LDX		O+18	W = O
E923	FF	042A	STX		W+18	
E926	F6	0444	LDA	B	O+18+2	
E929	F7	042C	STA	B	W+18+2	
E92C	FE	045A	LDX		P+18	O = P
E92F	FF	0442	STX		O+18	
E932	F6	045C	LDA	B	P+18+2	
E935	F7	0444	STA	B	O+18+2	

E938	FE	0472	LDX	Q+18	Y = Q
E93B	FF	04A2	STX	Y+18	
E93E	F6	0474	LDA B	Q+18+2	
E941	F7	04A4	STA B	Y+18+2	
E944	FE	048A	LDX	R+18	Q = R
E947	FF	0472	STX	Q+18	
E94A	F6	048C	LDA B	R+18+2	
E94D	F7	0474	STA B	Q+18+2	
E950	FE	0445	LDX	O+21	W = O
E953	FF	042D	STX	W+21	
E956	F6	0447	LDA B	O+21+2	
E959	F7	042F	STA B	W+21+2	
E95C	FE	045D	LDX	P+21	O = P
E95F	FF	0445	STX	O+21	
E962	F6	045F	LDA B	P+21+2	
E965	F7	0447	STA B	O+21+2	
E968	FE	0475	LDX	Q+21	Y = Q
E96B	FF	04A5	STX	Y+21	
E96E	F6	0477	LDA B	Q+21+2	
E971	F7	04A7	STA B	Y+21+2	
E974	FE	048D	LDX	R+21	Q = R
E977	FF	0475	STX	Q+21	
E97A	F6	048F	LDA B	R+21+2	
E97D	F7	0477	STA B	Q+21+2	
E980	CE	0000	LDX	#\$0000	
E983	5F		CLR B		
E984	FF	0448	STX	P+0	P = 0
E987	F7	044A	STA B	P+0+2	
E98A	FF	0478	STX	R+0	R = 0
E98D	F7	047A	STA B	R+0+2	
E990	FF	044B	STX	P+3	P = 3
E993	F7	044D	STA B	P+3+2	
E996	FF	047B	STX	R+3	R = 3
E999	F7	047D	STA B	R+3+2	
E99C	FF	044E	STX	P+6	P = 6
E99F	F7	0450	STA B	P+6+2	
E9A2	FF	047E	STX	R+6	R = 6
E9A5	F7	0480	STA B	R+6+2	
E9A8	FF	0451	STX	P+9	P = 9
E9AB	F7	0453	STA B	P+9+2	
E9AE	FF	0481	STX	R+9	R = 9
E9B1	F7	0483	STA B	R+9+2	
E9B4	FF	0454	STX	P+12	P = 12
E9B7	F7	0456	STA B	P+12+2	
E9BA	FF	0484	STX	R+12	R = 12
E9BD	F7	0486	STA B	R+12+2	
E9C0	FF	0457	STX	P+15	P = 15
E9C3	F7	0459	STA B	P+15+2	
E9C6	FF	0487	STX	R+15	R = 15
E9C9	F7	0489	STA B	R+15+2	
E9CC	FF	045A	STX	P+18	P = 18
E9CF	F7	045C	STA B	P+18+2	
E9D2	FF	048A	STX	R+18	R = 18
E9D5	F7	048C	STA B	R+18+2	

E9D8 FF 045D	STX	P+21	P = 21
E9DB F7 045F	STA B	P+21+2	
E9DE FF 048D	STX	R+21	R = 21
E9E1 F7 048F	STA B	R+21+2	
*			
E9E4 BD F3DF	JSR	MF21	W = W + S*A0
E9E7 0400	FDB	S+0, A0+0, W+0	
E9E9 04B8			
E9EB 0418			
E9ED BD F3E8	JSR	MF22	O = O + S*A1
E9F0 04C8	FDB	A1+0, O+0	
E9F2 0430			
E9F4 BD F3E8	JSR	MF22	P = P + S*A2
E9F7 04D8	FDB	A2+0, P+0	
E9F9 0448			
*			
E9FB BD F3DF	JSR	MF21	O = O + W*(B1+1)
E9FE 0418	FDB	W+0, B1+0, O+0	
EA00 04E8			
EA02 0430			
EA04 F6 041A	LDA B	W+0+2	
EA07 FB 0432	ADD B	O+0+2	
EA0A F7 0432	STA B	O+0+2	
EA0D F6 0419	LDA B	W+0+1	
EA10 F9 0431	ADC B	O+0+1	
EA13 B6 0418	LDA A	W+0	
EA16 B9 0430	ADC A	O+0	
EA19 B7 0430	STA A	O+0	
EA1C F7 0431	STA B	O+0+1	
EA1F BD F3E8	JSR	MF22	P = P + W*B2
EA22 04F8	FDB	B2+0, P+0	
EA24 0448			
EA26 BD F3E8	JSR	MF22	Y = Y + W*C0
EA29 0508	FDB	C0+0, Y+0	
EA2B 0490			
EA2D BD F3E8	JSR	MF22	Q = Q + W*C1
EA30 0518	FDB	C1+0, Q+0	
EA32 0460			
EA34 BD F3E8	JSR	MF22	R = R + W*C2
EA37 0528	FDB	C2+0, R+0	
EA39 0478			
*			
EA3B BD F3DF	JSR	MF21	Q = Q + Y*(D1+1)
EA3E 0490	FDB	Y+0, D1+0, Q+0	
EA40 0538			
EA42 0460			
EA44 F6 0492	LDA B	Y+0+2	
EA47 FB 0462	ADD B	Q+0+2	
EA4A F7 0462	STA B	Q+0+2	
EA4D F6 0491	LDA B	Y+0+1	
EA50 F9 0461	ADC B	Q+0+1	
EA53 B6 0490	LDA A	Y+0	
EA56 B9 0460	ADC A	Q+0	
EA59 B7 0460	STA A	Q+0	

EA5C F7 0461	STA B Q+0+1
EA5F BD F3E8	JSR MF22 $R = R + Y * D2$
EA62 0548	FDB D2+0, R+0
EA64 0478	
*	
EA66 BD F3DF	JSR MF21 $W = W + S * A0$
EA69 0403	FDB S+3, A0+2, W+3
EA6B 04BA	
EA6D 041B	
EA6F BD F3E8	JSR MF22 $O = O + S * A1$
EA72 04CA	FDB A1+2, O+3
EA74 0433	
EA76 BD F3E8	JSR MF22 $P = P + S * A2$
EA79 04DA	FDB A2+2, P+3
EA7B 044B	
*	
EA7D BD F3DF	JSR MF21 $O = O + W * (B1+1)$
EA80 041B	FDB W+3, B1+2, O+3
EA82 04EA	
EA84 0433	
EA86 F6 041D	LDA B W+3+2
EA89 FB 0435	ADD B O+3+2
EA8C F7 0435	STA B O+3+2
EA8F F6 041C	LDA B W+3+1
EA92 F9 0434	ADC B O+3+1
EA95 B6 041B	LDA A W+3
EA98 B9 0433	ADC A O+3
EA9B B7 0433	STA A O+3
EA9E F7 0434	STA B O+3+1
EAA1 BD F3E8	JSR MF22 $P = P + W * B2$
EAA4 04FA	FDB B2+2, P+3
EAA6 044B	
EAA8 BD F3E8	JSR MF22 $Y = Y + W * C0$
EAAB 050A	FDB C0+2, Y+3
EAAD 0493	
EAAF BD F3E8	JSR MF22 $Q = Q + W * C1$
EAB2 051A	FDB C1+2, Q+3
EAB4 0463	
EAB6 BD F3E8	JSR MF22 $R = R + W * C2$
EAB9 052A	FDB C2+2, R+3
EABB 047B	
*	
EABD BD F3DF	JSR MF21 $Q = Q + Y * (D1+1)$
EAC0 0493	FDB Y+3, D1+2, Q+3
EAC2 053A	
EAC4 0463	
EAC6 F6 0495	LDA B Y+3+2
EAC9 FB 0465	ADD B Q+3+2
EACC F7 0465	STA B Q+3+2
EACF F6 0494	LDA B Y+3+1
EAD2 F9 0464	ADC B Q+3+1
EAD5 B6 0493	LDA A Y+3
EAD8 B9 0463	ADC A Q+3
EADB B7 0463	STA A Q+3

EADE F7 0464	STA B Q+3+1	
EAE1 BD F3E8	JSR MF22	$R = R + Y * D2$
EAE4 054A	FDB D2+2, R+3	
EAE6 047B		
*		
EAE8 BD F3DF	JSR MF21	$W = W + S * A0$
EAEB 0406	FDB S+6, A0+4, W+6	
EAED 04BC		
EAEF 041E		
EAFF BD F3E8	JSR MF22	$O = O + S * A1$
EAFF 04CC	FDB A1+4, O+6	
EAFF 0436		
EAFF BD F3E8	JSR MF22	$P = P + S * A2$
EAFF 04DC	FDB A2+4, P+6	
EAFF 044E		
*		
EAFF BD F3DF	JSR MF21	$O = O + W * (B1+1)$
EB02 041E	FDB W+6, B1+4, O+6	
EB04 04EC		
EB06 0436		
EB08 F6 0420	LDA B W+6+2	
EB0B FB 0438	ADD B O+6+2	
EB0E F7 0438	STA B O+6+2	
EB11 F6 041F	LDA B W+6+1	
EB14 F9 0437	ADC B O+6+1	
EB17 B6 041E	LDA A W+6	
EB1A B9 0436	ADC A O+6	
EB1D B7 0436	STA A O+6	
EB20 F7 0437	STA B O+6+1	
EB23 BD F3E8	JSR MF22	$P = P + W * B2$
EB26 04FC	FDB B2+4, P+6	
EB28 044E		
EB2A BD F3E8	JSR MF22	$Y = Y + W * C0$
EB2D 050C	FDB C0+4, Y+6	
EB2F 0496		
EB31 BD F3E8	JSR MF22	$Q = Q + W * C1$
EB34 051C	FDB C1+4, Q+6	
EB36 0466		
EB38 BD F3E8	JSR MF22	$R = R + W * C2$
EB3B 052C	FDB C2+4, R+6	
EB3D 047E		
*		
EB3F BD F3DF	JSR MF21	$Q = Q + Y * (D1+1)$
EB42 0496	FDB Y+6, D1+4, Q+6	
EB44 053C		
EB46 0466		
EB48 F6 0498	LDA B Y+6+2	
EB4B FB 0468	ADD B Q+6+2	
EB4E F7 0468	STA B Q+6+2	
EB51 F6 0497	LDA B Y+6+1	
EB54 F9 0467	ADC B Q+6+1	
EB57 B6 0496	LDA A Y+6	
EB5A B9 0466	ADC A Q+6	
EB5D B7 0466	STA A Q+6	

EB60 F7 0467	STA B Q+6+1
EB63 BD F3E8	JSR MF22 R = R + Y*D2
EB66 054C	FDB D2+4, R+6
EB68 047E	
*	
EB6A BD F3DF	JSR MF21 W = W + S*A0
EB6D 0409	FDB S+9, A0+6, W+9
EB6F 04BE	
EB71 0421	
EB73 BD F3E8	JSR MF22 O = O + S*A1
EB76 04CE	FDB A1+6, O+9
EB78 0439	
EB7A BD F3E8	JSR MF22 P = P + S*A2
EB7D 04DE	FDB A2+6, P+9
EB7F 0451	
*	
EB81 BD F3DF	JSR MF21 O = O + W*(B1+1)
EB84 0421	FDB W+9, B1+6, O+9
EB86 04EE	
EB88 0439	
EB8A F6 0423	LDA B W+9+2
EB8D FB 043B	ADD B O+9+2
EB90 F7 043B	STA B O+9+2
EB93 F6 0422	LDA B W+9+1
EB96 F9 043A	ADC B O+9+1
EB99 B6 0421	LDA A W+9
EB9C B9 0439	ADC A O+9
EB9F B7 0439	STA A O+9
EBA2 F7 043A	STA B O+9+1
EBA5 BD F3E8	JSR MF22 P = P + W*B2
EBA8 04FE	FDB B2+6, P+9
EBA A 0451	
EBAC BD F3E8	JSR MF22 Y = Y + W*C0
EBAF 050E	FDB C0+6, Y+9
EBB1 0499	
EBB3 BD F3E8	JSR MF22 Q = Q + W*C1
EBB6 051E	FDB C1+6, Q+9
EBB8 0469	
EBBA BD F3E8	JSR MF22 R = R + W*C2
EBBD 052E	FDB C2+6, R+9
EBBF 0481	
*	
EBC1 BD F3DF	JSR MF21 Q = Q + Y*(D1+1)
EBC4 0499	FDB Y+9, D1+6, Q+9
EBC6 053E	
EBC8 0469	
EBCA F6 049B	LDA B Y+9+2
EBCD FB 046B	ADD B Q+9+2
EBD0 F7 046B	STA B Q+9+2
EBD3 F6 049A	LDA B Y+9+1
EBD6 F9 046A	ADC B Q+9+1
EBD9 B6 0499	LDA A Y+9
EBDC B9 0469	ADC A Q+9
EBDF B7 0469	STA A Q+9

EBE2 F7 046A	STA B	Q+9+1	
EBE5 BD F3E8	JSR	MF22	R = R + Y*D2
EBE8 054E	FDB	D2+6, R+9	
EBEA 0481			
*			
EBEC BD F3DF	JSR	MF21	W = W + S*A0
EBEF 040C	FDB	S+12, A0+8, W+12	
EBF1 04C0			
EBF3 0424			
EBF5 BD F3E8	JSR	MF22	O = O + S*A1
EBF8 04D0	FDB	A1+8, O+12	
EBFA 043C			
EBFC BD F3E8	JSR	MF22	P = P + S*A2
EBFF 04E0	FDB	A2+8, P+12	
EC01 0454			
*			
EC03 BD F3DF	JSR	MF21	O = O + W*(B1+1)
EC06 0424	FDB	W+12, B1+8, O+12	
EC08 04F0			
EC0A 043C			
EC0C F6 0426	LDA B	W+12+2	
EC0F FB 043E	ADD B	O+12+2	
EC12 F7 043E	STA B	O+12+2	
EC15 F6 0425	LDA B	W+12+1	
EC18 F9 043D	ADC B	O+12+1	
EC1B B6 0424	LDA A	W+12	
EC1E B9 043C	ADC A	O+12	
EC21 B7 043C	STA A	O+12	
EC24 F7 043D	STA B	O+12+1	
EC27 BD F3E8	JSR	MF22	P = P + W*B2
EC2A 0500	FDB	B2+8, P+12	
EC2C 0454			
EC2E BD F3E8	JSR	MF22	Y = Y + W*C0
EC31 0510	FDB	C0+8, Y+12	
EC33 049C			
EC35 BD F3E8	JSR	MF22	Q = Q + W*C1
EC38 0520	FDB	C1+8, Q+12	
EC3A 046C			
EC3C BD F3E8	JSR	MF22	R = R + W*C2
EC3F 0530	FDB	C2+8, R+12	
EC41 0484			
*			
EC43 BD F3DF	JSR	MF21	Q = Q + Y*(D1+1)
EC46 049C	FDB	Y+12, D1+8, Q+12	
EC48 0540			
EC4A 046C			
EC4C F6 049E	LDA B	Y+12+2	
EC4F FB 046E	ADD B	Q+12+2	
EC52 F7 046E	STA B	Q+12+2	
EC55 F6 049D	LDA B	Y+12+1	
EC58 F9 046D	ADC B	Q+12+1	
EC5B B6 049C	LDA A	Y+12	
EC5E B9 046C	ADC A	Q+12	
EC61 B7 046C	STA A	Q+12	

EC64 F7 046D	STA B Q+12+1
EC67 BD F3E8	JSR MF22 R = R + Y*D2
EC6A 0550	FDB D2+8, R+12
EC6C 0484	
*	
EC6E BD F3DF	JSR MF21 W = W + S*A0
EC71 040F	FDB S+15, A0+10, W+15
EC73 04C2	
EC75 0427	
EC77 BD F3E8	JSR MF22 O = O + S*A1
EC7A 04D2	FDB A1+10, O+15
EC7C 043F	
EC7E BD F3E8	JSR MF22 P = P + S*A2
EC81 04E2	FDB A2+10, P+15
EC83 0457	
*	
EC85 BD F3DF	JSR MF21 O = O + W*(B1+1)
EC88 0427	FDB W+15, B1+10, O+15
EC8A 04F2	
EC8C 043F	
EC8E F6 0429	LDA B W+15+2
EC91 FB 0441	ADD B O+15+2
EC94 F7 0441	STA B O+15+2
EC97 F6 0428	LDA B W+15+1
EC9A F9 0440	ADC B O+15+1
EC9D B6 0427	LDA A W+15
ECA0 B9 043F	ADC A O+15
ECA3 B7 043F	STA A O+15
ECA6 F7 0440	STA B O+15+1
ECA9 BD F3E8	JSR MF22 P = P + W*B2
ECAC 0502	FDB B2+10, P+15
ECAE 0457	
ECB0 BD F3E8	JSR MF22 Y = Y + W*C0
ECB3 0512	FDB C0+10, Y+15
ECB5 049F	
ECB7 BD F3E8	JSR MF22 Q = Q + W*C1
ECBA 0522	FDB C1+10, Q+15
ECBC 046F	
ECBE BD F3E8	JSR MF22 R = R + W*C2
ECC1 0532	FDB C2+10, R+15
ECC3 0487	
*	
ECC5 BD F3DF	JSR MF21 Q = Q + Y*(D1+1)
ECC8 049F	FDB Y+15, D1+10, Q+15
ECCA 0542	
ECCC 046F	
ECCE F6 04A1	LDA B Y+15+2
ECD1 FB 0471	ADD B Q+15+2
ECD4 F7 0471	STA B Q+15+2
ECD7 F6 04A0	LDA B Y+15+1
ECDA F9 0470	ADC B Q+15+1
ECDD B6 049F	LDA A Y+15
ECE0 B9 046F	ADC A Q+15
ECE3 B7 046F	STA A Q+15

ECE6 F7 0470
ECE9 BD F3E8
ECEC 0552
ECEE 0487

*

ECF0 BD F3DF
ECF3 0412
ECF5 04C4
ECF7 042A
ECF9 BD F3E8
ECFC 04D4
ECFE 0442
ED00 BD F3E8
ED03 04E4
ED05 045A

*

ED07 BD F3DF
ED0A 042A
ED0C 04F4
ED0E 0442
ED10 F6 042C
ED13 FB 0444
ED16 F7 0444
ED19 F6 042B
ED1C F9 0443
ED1F B6 042A
ED22 B9 0442
ED25 B7 0442
ED28 F7 0443
ED2B BD F3E8
ED2E 0504
ED30 045A
ED32 BD F3E8
ED35 0514
ED37 04A2
ED39 BD F3E8
ED3C 0524
ED3E 0472
ED40 BD F3E8
ED43 0534
ED45 048A

*

ED47 BD F3DF
ED4A 04A2
ED4C 0544
ED4E 0472
ED50 F6 04A4
ED53 FB 0474
ED56 F7 0474
ED59 F6 04A3
ED5C F9 0473
ED5F B6 04A2
ED62 B9 0472
ED65 B7 0472

STA B Q+15+1
JSR MF22 R = R + Y*D2
FDB D2+10, R+15

JSR MF21 W = W + S*A0
FDB S+18, A0+12, W+18

JSR MF22 O = O + S*A1
FDB A1+12, O+18

JSR MF22 P = P + S*A2
FDB A2+12, P+18

JSR MF21 O = O + W*(B1+1)
FDB W+18, B1+12, O+18

LDA B W+18+2

ADD B O+18+2

STA B O+18+2

LDA B W+18+1

ADC B O+18+1

LDA A W+18

ADC A O+18

STA A O+18

STA B O+18+1

JSR MF22 P = P + W*B2
FDB B2+12, P+18

JSR MF22 Y = Y + W*C0
FDB C0+12, Y+18

JSR MF22 Q = Q + W*C1
FDB C1+12, Q+18

JSR MF22 R = R + W*C2
FDB C2+12, R+18

JSR MF21 Q = Q + Y*(D1+1)
FDB Y+18, D1+12, Q+18

LDA B Y+18+2

ADD B Q+18+2

STA B Q+18+2

LDA B Y+18+1

ADC B Q+18+1

LDA A Y+18

ADC A Q+18

STA A Q+18

ED68 F7 0473	STA B	Q+18+1
ED6B BD F3E8	JSR	MF22 R = R + Y*D2
ED6E 0554	FDB	D2+12, R+18
ED70 048A		
*		
ED72 BD F3DF	JSR	MF21 W = W + S*A0
ED75 0415	FDB	S+21, A0+14, W+21
ED77 04C6		
ED79 042D		
ED7B BD F3E8	JSR	MF22 O = O + S*A1
ED7E 04D6	FDB	A1+14, O+21
ED80 0445		
ED82 BD F3E8	JSR	MF22 P = P + S*A2
ED85 04E6	FDB	A2+14, P+21
ED87 045D		
*		
ED89 BD F3DF	JSR	MF21 O = O + W*(B1+1)
ED8C 042D	FDB	W+21, B1+14, O+21
ED8E 04F6		
ED90 0445		
ED92 F6 042F	LDA B	W+21+2
ED95 FB 0447	ADD B	O+21+2
ED98 F7 0447	STA B	O+21+2
ED9B F6 042E	LDA B	W+21+1
ED9E F9 0446	ADC B	O+21+1
EDA1 B6 042D	LDA A	W+21
EDA4 B9 0445	ADC A	O+21
EDA7 B7 0445	STA A	O+21
EDAA F7 0446	STA B	O+21+1
EDAD BD F3E8	JSR	MF22 P = P + W*B2
EDB0 0506	FDB	B2+14, P+21
EDB2 045D		
EDB4 BD F3E8	JSR	MF22 Y = Y + W*C0
EDB7 0516	FDB	C0+14, Y+21
EDB9 04A5		
EDBB BD F3E8	JSR	MF22 Q = Q + W*C1
EDBE 0526	FDB	C1+14, Q+21
EDC0 0475		
EDC2 BD F3E8	JSR	MF22 R = R + W*C2
EDC5 0536	FDB	C2+14, R+21
EDC7 048D		
*		
EDC9 BD F3DF	JSR	MF21 Q = Q + Y*(D1+1)
EDCC 04A5	FDB	Y+21, D1+14, Q+21
EDCE 0546		
EDD0 0475		
EDD2 F6 04A7	LDA B	Y+21+2
EDD5 FB 0477	ADD B	Q+21+2
EDD8 F7 0477	STA B	Q+21+2
EDDB F6 04A6	LDA B	Y+21+1
EDDE F9 0476	ADC B	Q+21+1
EDE1 B6 04A5	LDA A	Y+21
EDE4 B9 0475	ADC A	Q+21
EDE7 B7 0475	STA A	Q+21


```

EDEA F7 0476          STA B   Q+21+1
EDED BD F3E8          JSR     MF22      R = R + Y*D2
EDF0 0556             FDB     D2+14,R+21
EDF2 048D
*
EDF4 39               RTS
*   15.457 M SEC.
**  END   **
*
*
EDF5      WBL      EQU * WRITE BOOT LOOP ROUTINE
*
* - THIS ROUTINE IS USED TO REWRITE THE BOOT LOOP
*   IN THE BUBBLE MEMORY SYSTEM
* - IF THE BOOT LOOP DOES NOT CONTAIN THE DATA
*   IN TABLE "BLTAB", THE BUBBLE MEMORY SYSTEM
*   WILL NOT WORK
* - THE "BPK - 72 USER'S MANUAL" EXPLAINS WHAT
*   FUNCTIONS MUST BE EXECUTED IN ORDER TO CHECK
*   THE DATA IN THE BOOT LOOP. THESES FUNCTIONS
*   ARE DESCRIBED IN ROM6 AND ARE EXECUTED VIA THE
*   DEBUG SECTION OF THE MONITOR.
* - THE DATA IN THE BOOT LOOP CAN BE ERRONEOUSLY
*   CHANGED ONLY IF THE BMC CHIP GOES "BEZIRK"
*   DUE TO NOISE ON THE POWER SUPPLY LINES.
* - THIS ROUTINE REQUIRES NO INPUTS, PRODUCES NO
*   OUTPUTS, AND CAN BE EXECUTED USING THE "SXVM"
*   ROUTINE IN ROM7 IF THE  DEBUG SECTION OF THE
*   MONITOR IS ENABLED
* - THE EXECUTION OF THIS ROUTINE IS SUCCESSFUL
*   IF THE KEY STROKE SEQUENCE," (.),(.BDB),(0),
*   (0)", PRODUCES THE DISPLAY "B00  40  "
*
EDF5 86 19             LDA A   #$19      SEND "ABORT"
EDF7 B7 AC01           STA A   BCSR
EDFA B6 AC01 WBL1      LDA A   BCSR      WAIT FOR
EDFD 81 40             CMP A   #$40      "OP. COMPLETE"
EDFF 26 F9             BNE     WBL1
*
EE01 86 1E             LDA A   #$1E      SEND "PURGE"
EE03 B7 AC01           STA A   BCSR
EE06 B6 AC01 WBL2      LDA A   BCSR      WAIT FOR
EE09 81 40             CMP A   #$40      "OP. COMPLETE"
EE0B 26 F9             BNE     WBL2
*
EE0D 86 1D             LDA A   #$1D      SEND "FIFO RESET"
EE0F B7 AC01           STA A   BCSR
EE12 B6 AC01 WBL3      LDA A   BCSR      WAIT FOR
EE15 81 40             CMP A   #$40      "OP. COMPLETE"
EE17 26 F9             BNE     WBL3
*
EE19 86 0B             LDA A   #$0B      WRITE TO BMC REG.
EE1B B7 AC01           STA A   BCSR
EE1E 86 01             LDA A   #$01      BLR LSB

```



```

EE20 B7 AC00      STA A   BCDR
EE23 86 10      LDA A   #$10      BLR MSB
EE25 B7 AC00      STA A   BCDR
EE28 86 10      LDA A   #$10      ER
EE2A B7 AC00      STA A   BCDR
EE2D 86 00      LDA A   #$00      AR LSB
EE2F B7 AC00      STA A   BCDR
EE32 86 00      LDA A   #$00      AR MSB
EE34 B7 AC00      STA A   BCDR
EE37 01          NOP
EE38 01          NOP
EE39 01          NOP
*
EE3A C6 28      LDA B   #40      WRITE $FF 40 TIMES
EE3C 86 FF      LDA A   #$FF      TO FIFO
EE3E B7 AC00    WBLL4  STA A   BCDR
EE41 5A          DEC B
EE42 26 FA      BNE      WBLL4
*
EE44 86 16      LDA A   #$16      SEND "WRITE BOOT
EE46 B7 AC01      STA A   BCSR      LOOP REGISTER"
EE49 B6 AC01    WBLL5  LDA A   BCSR      WAIT FOR
EE4C 81 40      CMP A   #$40      "OP. COMPLETE"
EE4E 26 F9      BNE      WBLL5
*
EE50 CE EE64      LDX      #BLTAB      WRITE BOOT LOOP
EE53 C6 29      LDA B   #41      DATA AND A ZERO
EE55 A6 00      WBLL6  LDA A   0,X      BYTE TO FIFO
EE57 B7 AC00      STA A   BCDR
EE5A 08          INX
EE5B 5A          DEC B
EE5C 26 F7      BNE      WBLL6
*
EE5E 86 17      LDA A   #$17      SEND "WRITE BOOT
EE60 B7 AC01      STA A   BCSR      LOOP"
*
EE63 39          RTS      RETURN
*
**   END   **
*
*
EE64      BLTAB   EQU      *      BOOT LOOP TABLE
*
* - THIS TABLE CONTAINS THE DATA THAT SHOULD
*   BE STORED WITHIN THE BOOT LOOP
* - THIS DATA IS ALSO PRINTED ON THE PACKAGE
*   OF THE BUBBLE MEMORY DEVICE
*
EE64 FF      FCB      $FF,$FF,$FF,$E7,$FF,$DF,$FD,$FF
EE65 FF
EE66 FF
EE67 E7
EE68 FF
EE69 DF

```



```

EE6A FD
EE6B FF
EE6C 7F      FCB      $7F,$F7,$DF,$DD,$EF,$FF,$FF,$FF
EE6D F7
EE6E DF
EE6F DD
EE70 EF
EE71 FF
EE72 FF
EE73 FF
EE74 FF      FCB      $FF,$FF,$FF,$0F,$FF,$FF,$FF,$EF
EE75 FF
EE76 FF
EE77 0F
EE78 FF
EE79 FF
EE7A FF
EE7B EF
EE7C FE      FCB      $FE,$FF,$EF,$BF,$FF,$FF,$F7,$FF
EE7D FF
EE7E EF
EE7F BF
EE80 FF
EE81 FF
EE82 F7
EE83 FF
EE84 EF      FCB      $EF,$FF,$FF,$FF,$EB,$77,$35,$57
EE85 FF
EE86 FF
EE87 FF
EE88 EB
EE89 77
EE8A 35
EE8B 57
EE8C 00      FCB      $00

```

*

*

*

*

* JUMP TABLE FOR ROM 5

*

```

EFD0          ORG      $EFD0
EFD0 7E E800   JMP      FILT
EFD3 7E EDF5   JMP      WBL
                END

```


ROM6

M68SAM is the property of Motorola Spd, Inc.
Copyright 1974 by Motorola Inc.

Motorola M6800 Cross Assembler, Release 1.1

NAM ROM6 MISSALANEOUS ROUTINES

```

*
*
* BUBBLE MEMORY CONTROLLER ADDRESSES
*
AC01      BCSR      EQU $AC01 - BUBBLE CONTROL/STATUS REG.
AC00      BCDR      EQU $AC00 - BUBBLE CONTROLER DATA REG.
005B      BMOBS     EQU $005B
005A      BMIBS     EQU $005A
006F      BMOBC     EQU $006F
0057      BMRA      EQU $0057
0055      BMWA      EQU $0055
0059      BSS       EQU $0059
001E      BUF2SR    EQU $001E
07FD      BMSA      EQU $07FD
0650      BSIL      EQU $0650
0091      BSOLF     EQU $0091
*
*
* TIMER ADDRESSES ( MC6840 )
*
7C06      TL3H      EQU $7C06 - TIMER # 1 LATCHES
7C04      TL2H      EQU $7C04
7C02      TL1H      EQU $7C02
7C00      CR3       EQU $7C00 - # 1 CONTROL REG.;WRITE ONLY.
7C01      CR2       EQU $7C01
7C00      CR1       EQU $7C00
7C01      TSTATR    EQU $7C01 - # 1 STATUS REG.;READ ONLY.
9406      TL6H      EQU $9406 - TIMER # 2 LATCHES
9404      TL5H      EQU $9404
9402      TL4H      EQU $9402
9400      CR6       EQU $9400 - # 2 CONTROL REG.;WRITE ONLY.
9401      CR5       EQU $9401
9400      CR4       EQU $9400
9401      TSTR2     EQU $9401 - # 2 STATUS REG.;READ ONLY.
*
*
* PROGRAMMABLE KEYBOARD DISPLAY INTERFACE (I8279-5)
*
8001      PKDICR    EQU $8001 - CONTROL REGISTER;WRITE ONLY
8001      PKDISR    EQU $8001 - STATUS REGISTER;READ ONLY
8000      FIFO      EQU $8000 - KEYBOARD INPUT RAM;READ ONLY
8000      DISRAM    EQU $8000 - DISPLAY RAM; READ/WRITE
*
*
* SUBROUTINE LABLES
*

```



```

FFCA    DSPSN    EQU    $FFCA
FFCD    DSPDP    EQU    $FFCD
FFD0    CLDISP   EQU    $FFD0
FFD6    DTSR2    EQU    $FFD6
FFDF    DSPXAB   EQU    $FFDF
FFE2    DSTOBI   EQU    $FFE2
FFC1    READ2    EQU    $FFC1
FFBB    WRITE0    EQU    $FFBB
FFC4    WRITE2    EQU    $FFC4
FFE8    READ4    EQU    $FFE8
FFEB    WRITE4    EQU    $FFEB
FFF1    DELAY2    EQU    $FFF1
FBF6    SCNDSR   EQU    $FBF6
FFC7    RDWL     EQU    $FFC7
FFBE    FLSH     EQU    $FFBE
F3F1    TMSR     EQU    $F3F1
F3DF    MF21     EQU    $F3DF
F3E8    MF22     EQU    $F3E8
F7E3    TX1CHS   EQU    $F7E3
F7F5    TX2CHS   EQU    $F7F5
B7E6    BMSI     EQU    $B7E6
F3D6    SAMP     EQU    $F3D6
F3E5    TMHB     EQU    $F3E5
F3EB    CKSM     EQU    $F3EB
F3EE    GTXB     EQU    $F3EE
EFD0    FILT     EQU    $EFD0
F3F4    MDTI     EQU    $F3F4

```

*

*

*

LABLES FOR KEYBOARD ROUTINES

*

```

F7E6    ERROR    EQU    $F7E6
BFDF    SIG      EQU    $BFDF

```

*

*

*

OTHER LABLES

*

```

B800    RM8RST   EQU    $B800

```

*

*

*

LINKAGE TABLE ENTRIES FOR SWI ROUTINES

*

```

F400    IERR     EQU    $F400
F402    ISTX1    EQU    $F402
F404    ISTX2    EQU    $F404
F406    ISABX    EQU    $F406
F408    ILABX    EQU    $F408
F40A    IABTX    EQU    $F40A
F40C    IAITX    EQU    $F40C
F40E    IRX2     EQU    $F40E
F410    IPSHX    EQU    $F410

```

*

*

*

MONITOR GLOBAL VARIABLES AND FLAGS


```

*
0000    READEN EQU $0000 - READ ENABLE
0004    ENTRY1 EQU $0004 - ENTRIES VIA ENTR FUNCTION
0005    ENTRY2 EQU $0005
0006    ENTRY3 EQU $0006
0007    ENTRY4 EQU $0007
0008    ENTRY5 EQU $0008
0009    ENTRY6 EQU $0009
000A    ENTRY7 EQU $000A
000B    ENTRY8 EQU $000B
000D    IXRV    EQU $000D,E  X REG. STORAGE, MAIN LINES
0010    MONFLG EQU $0010 - IND. MAIN STOPED BY MONITOR
0011    SECFLG EQU $0011 - IND. 2ND KEYBRD. FTN. SEL.
0012    LEDNO   EQU $0012 - LED TO BE WRITTEN NEXT

```

```

*          INTO NEXT, (FIRST LED = #0)
*
*
*

```

```

*  RAM LOCATIONS USED ONLY BY ROM6
*

```

```

004A          ORG    $004A
0600    FIFOI EQU $0600 - FIFO INPUT STORAGE
0628    FIFOO EQU $0628 - FIFO OUTPUT STORAGE

```

```

*  OTHER VARIABLES USED BY ROM 6
*

```

```

0042          ORG    $0042
0042 0001    FLGC    RMB 1
0043 0002          RMB 2
0045 0004          RMB 4
0049 0001    RXD     RMB 1
        0560    TXBLK EQU $0560
        0490    Y      EQU $0490
        0400    S      EQU $0400
        0086    SYST   EQU $0086
        008B    SELFLG EQU $008B
        0089    FTNFLG EQU $0089
        0032    SPLVEC EQU $0032
        0085    SPLCRS EQU $0085
        0087    MLAC   EQU $0087

```

```

*  LOCATIONS OF SOME TABLES USED BY ROM5
*

```

```

F780    EOFTBL EQU $F780
F78C    PTIPON EQU $F78C
F798    PTIPOF EQU $F798
F7A5    ENDTBL EQU $F7A5
F423    ERRORT EQU $F423
BF58    DUSEHS EQU $BF58
BF28    DUSEL  EQU $BF28
BF20    DBMSFL EQU $BF20
E740    D104TB EQU $E740

```



```

*
*
*   SOME CONSTANTS USED BY ROM 6
*
0080      FRFR      EQU $80
0081      FREN      EQU $81
*
*
*****
*
E000          ORG      $E000
*              *****
*
*
*
E000      RM6RST EQU      *              ROM 6   RESET
*
*
E000 7E B800          JMP      RM8RST
E003 CE E777 RM6WAI LDX      #RDYTB L
E006 BD FFD6          JSR      DTSR2
E009 0E              CLI
E00A 3E              RSTWAI WAI
E00B 20 FD          BRA      RSTWAI      STOP IN A WAIT LOOP
**      END      **
*
*
E00D      TIMT      EQU      *              TIMER TEST
*
* - 'A MAINLINE ROUTINE TO TEST THE TIME CLOCK, PTM #2
* - DISPLAYS THE TIME EVERY .1 SEC.
* - TIME IS DISPLAYED AS      HHMMSS
* - AN INFINITE LOOP
*
E00D 36          TIMTL1 PSH A          *** MAKE SPACE
E00E 36          PSH A          ** FOR TIME BYTES
E00F 36          PSH A          *
E010 BD F3F1          JSR      TMSR      GET HH,MM,SS
E013 BD FFD0          JSR      CLDISP
E016 86 04          LDA A      #$04
E018 BD FFBB          JSR      WRITE0
E01B 32          PUL A          * LOAD SS
E01C BD FFC4          JSR      WRITE2      DISPLAY SS
E01F 4F          CLR A
E020 BD FFBB          JSR      WRITE0
E023 33          PUL B          ** LOAD MM
E024 32          PUL A          ***LOAD HH
E025 BD FFEB          JSR      WRITE4      DISPLAY HH,MM
E028 C6 01          LDA B      #$01
E02A BD FFCD          JSR      DSPDP      HH,DEC. POINT,MM
E02D C6 03          LDA B      #$03
E02F BD FFCD          JSR      DSPDP      MM,DEC. POINT,SS
E032 BD FFF1          JSR      DELAY2

```



```

E035 0032          FDB      $0032
E037 20 D4          BRA      TIMTL1
*
**  END  **
*
*
E039 MF21T EQU * MULT. FRACTIONAL 2'S COMP. #1 TEST
*
* - ROUTINE TO TEST ROUTINE MF21
* - "ENTR" FUNCTION IS USED TO ENTER X AND Y, THE
*   FRACTIONAL 2'S COMPLEMENT NUMBERS TO BE MULTIPLIED
* - ENTER 1)ADDRESS OF TEST ROUTINE,2)"GOTO",3)"STRT"
*   AND THE 16 BIT PRODUCT IS DISPLAYED
* - A MAINLINE ROUTINE,  CHANGES IXRV
*
E039 CE 0000          LDX      #$0000      CLEAR PRODUCT LOC.
E03C DF 09          STX      ENTRY6
E03E 7F 000B          CLR      ENTRY8
E041 DE 06          LDX      ENTRY3      SHIFT MULTIPLIER
E043 DF 07          STX      ENTRY4      DOWN
E045 7F 0006          CLR      ENTRY3      MULT. CONTENTS OF
E048 BD F3DF          JSR      MF21      ENTRY1,ENTRY3
E04B 0004          FDB      ENTRY1,ENTRY4,ENTRY6
E04D 0007
E04F 0009
E051 BD FFD0          JSR      CLDISP      PROD. IN ENTRY6
E054 96 09          LDA      A      ENTRY6      DISPLAY PRODUCT
E056 D6 0A          LDA      B      ENTRY7
E058 BD FFEB          JSR      WRITE4
E05B 96 0B          LDA      A      ENTRY8
E05D BD FFC4          JSR      WRITE2
E060 7C 0010          INC      MONFLG      IND. MONITOR ACTIVE
E063 3E          MF21TW WAI      STOP
E064 20 FD          BRA      MF21TW
*
**  END  **
*
*
E066 MF22T EQU * MULT. FRACTIONAL 2'S COMP. #2 TEST
*
* - ROUTINE TO TEST ROUTINE MF22
* - "ENTR" FUNCTION IS USED TO ENTER Y, THE
*   FRACTIONAL 2'S COMPLEMENT NUMBER TO BE MULTIPLIED
*   WITH X ALREADY ENTERED INTO MULT. CHIP USING MF21T
* - ENTER 1)ADDRESS OF TEST ROUTINE,2)"GOTO",3)"STRT"
*   AND THE 16 BIT PRODUCT IS DISPLAYED
* - A MAINLINE ROUTINE,  CHANGES IXRV
*
E066 CE 0000          LDX      #$0000      CLEAR PRODUCT LOC.
E069 DF 09          STX      ENTRY6      MULT. CONTENTS OF X
E06B 7F 000B          CLR      ENTRY8
E06E 7F 0006          CLR      ENTRY3
E071 BD F3E8          JSR      MF22      WITH ENTRY1
E074 0004          FDB      ENTRY1,ENTRY6      PROD. IN IXRV

```



```

E076 0009
E078 BD FFD0      JSR      CLDISP
E07B 96 09        LDA A    ENTRY6      DISPLAY PRODUCT
E07D D6 0A        LDA B    ENTRY7
E07F BD FFEB      JSR      WRITE4
E082 96 0B        LDA A    ENTRY8
E084 BD FFC4      JSR      WRITE2
E087 7C 0010      INC      MONFLG      MONITOR ACTIVE
E08A 3E           MF22TW WAI          STOP
E08B 20 FD        BRA      MF22TW
*
**  END  **
*
*
E08D      CFOF      EQU      *          COPY/FORTRAN OFF
*
* - A SUBROUTINE TO STOP THE TRANSMISSION OF DATA
* - TO A COPY OR FORTRAN PROCESS IN MTS
* - DISABLES TX2,RX2 BY ENABLING TX1,RX1
* - CHANGES A,X
*
E08D 07          TPA
E08E 36          PSH A          *** SAVE INT. MASK
E08F 0E          CLI          ALLOW FOR TX INTS.
E090 CE F780     LDX      #EOFTBL  STOP COPY/FORT
E093 BD F7E3     JSR      TX1CHS  USE TX1,RX1
E096 BD FFF1     JSR      DELAY2
E099 00F9        FDB      $00F9   DELAY .5 SEC FOR TX
E09B CE F798     LDX      #PTIPOF  EXIT PTIP MODE (SEE
E09E BD F7E3     JSR      TX1CHS  MTS TERMINAL USERS
E0A1 BD FFF1     JSR      DELAY2  GUIDE)
E0A4 0030        FDB      $0030
E0A6 32          PUL A          *** RESTORE INT. MASK
E0A7 06          TAP
E0A8 39          RTS
*
**  END  **
*
*
E0A9 SPTP EQU *   SAVE PARTIALLY TRANSMITTED PAGES
*
* - THIS ROUTINE IS CALLED AFTER TRANSMISSION IS
* - STOPPED SO THAT ANY PARTIALLY TRANSMITTED PAGE
* - WILL BE SAVED IN THE BMS SO NO DATA IS LOST
* - AFTER THIS ROUTINE IS RUN AND TRANSMISSION
* - IS STARTED AGAIN, ONE OR TWO LINES MAY BE
* - TRANSMITTED A SECOND TIME
* - DOES NOT ALLOW INTERRUPTS AS THE STATUS OF THE BMS
* - IS NOT TO CHANGE DURING EXECUTION
* - TO BE USED AS PART OF A MAINLINE SO AS TO NOT
* - AFFECT ANY INTERRUPT DRIVEN TIME CRITICAL PROCESS
* - ( CONTAINS A WAIT LOOP )
* - NOT TO BE USED CONCURRENTLY WITH OR BEFORE "BMSO"
* - AS "BMSO" MAY REMOVE THE PAGE JUST SAVED

```


* - INPUTS : BMOBS,BMOBC,BUF2SR,BMWA,BMRA,BSS,BMIBS
 * - OUTPUTS: V IN CCR = SET IF DATA WAS LOST
 * - CHANGES V IN CCR,BMRA,BSS,BMOBS

```

*
EOA9 07          TPA
EOAA 36          PSH A          *** SAVE INT. MASK
EOAB 0E          CLI          ALLOW FOR BMS INTS.
*
EOAC D6 5B      SPTPL1 LDA B    BMOBS
EOAE 2B FC          BMI      SPTPL1  WAIT UNTIL READ IDLE
*
EOB0 0F          SEI          DO NOT ALLOW INT.
EOB1 7F 005B      CLR          BMOBS  CLEAR ANY READ REQ.
EOB4 4F          CLR A        # OF NON TXED BYTES
EOB5 D6 91      LDA B    BSOLF  ADD # BYTES IN BSOL
EOB7 27 02      BEQ      SPTPS0  SKIP IF ZERO
EOB9 8B 18      ADD A    #24
EOBB D6 1E      SPTPS0 LDA B    BUF2SR
EOBD C1 01      CMP B    #$01
EOBF 27 02      BEQ      SPTPS1  SKIP IF BUF2 EMPTY
EOC1 8B 18      ADD A    #24      1 LINE NOT TXED
EOC3 9B 6F      SPTPS1 ADD A    BMOBC  GET TOTAL # OF BYTES
EOC5 26 04      BNE      SPTPS2  SOME DATA NOT TXED?
*
EOC7 32          SPTPX1 PUL A          ***
EOC8 06          TAP          RESTORE INT. MASK
EOC9 0A          CLV          IND. NO DATA LOST
EOCA 39          RTS
*
EOCB 36          SPTPS2 PSH A          ** SAVE # OF BYTES
EOCC BD E0F4      JSR      SPSR      TRY TO SAVE A PAGE
EOCF 32          PUL A          ** RESTORE # OF BYTES
EOD0 29 09      BVS      SPTPXF  EXIT IF PG NOT SVD.
EOD2 80 40      SUB A    #64      EXIT IF A PAGE OR
EOD4 2F F1      BLE      SPTPX1  LESS NOT TXED
*
EOD6 BD E0F4      JSR      SPSR      SAVE ANOTHER PAGE?
EOD9 28 EC      BVC      SPTPX1  EXIT IF SAVED
*
EODB CE E738      SPTPXF LDX      #DDATAL  DISPLAY "DATA.LST"
EODE 86 07      LDA A    #$07
EOE0 5F          CLR B
EOE1 BD E5D8      JSR      DSCT
EOE4 32          PUL A          ***
EOE5 06          TAP          RESTORE INT. MASK
EOE6 0B          SEV          INDICATE DATA LOST
EOE7 39          RTS
*
** END **
EOE8 000C          RMB      12
*
*
EOF4 SPSR EQU *    SAVE PAGE SUBROUTINE
*
```



```

* - THIS ROUTINE IS CALLED BY "SPTP" TO SAVE
*   THE PAGE IN THE BMS THAT WAS MOST RECENTLY
*   READ ( PAGE # BMRA-1 )
* - INPUTS : BMRA,BMWA,BSS,BMIBS
* - OUTPUTS: V IN CCR SET IF A PAGE WAS NOT SAVED
*           BMWA,BSS
* - CHANGES B,X,CCR
* - ASSUMES THE READ PROCESS IS NOT ACTIVE
*   ( CASE 4 NOT POSSIBLE )
*

```

```

E0F4 D6 5A          LDA B  BMIBS
E0F6 C5 A0          BIT B  #$A0  IF WRITE NOT ACT. AND
E0F8 27 34          BEQ      SPSRC3 NOT REQ.,GO TO CASE 3
E0FA C8 A0          EOR B  #$A0
E0FC C5 A0          BIT B  #$A0  IF WRITE ACT. AND REQ.
E0FE 27 10          BEQ      SPSRC2  GO TO CASE 2
*

```

```

* CASE 1          * WRITE ACTIVE BUT NOT REQUESTED
*

```

```

E100 DE 57          LDX      BMRA      BMS FULL
E102 26 03          BNE      SPSRS4    IF BMRA-1=BMWA
E104 CE 07FD        LDX      #BMSA    DEC. WITH WRAP
E107 09             SPSRS4 DEX      AROUND
E108 9C 55          CPX      BMWA
E10A 27 3A          BEQ      SPSRXF    EXIT IF BMS FULL
E10C DF 57          STX      BMRA      SAVE PAGE
E10E 0A             CLV              INDICATE PAGE SAVED
E10F 39             RTS
*

```

```

* CASE2          * WRITE ACTIVE AND REQUESTED
*

```

```

E110 DE 57          SPSRC2 LDX      BMRA      BMS FULL
E112 26 03          BNE      SPSRS1    IF BMRA-2=BMWA
E114 CE 07FD        LDX      #BMSA    DEC. WITH WRAP
E117 09             SPSRS1 DEX      AROUND
E118 26 03          BNE      SPSRS2
E11A CE 07FD        LDX      #BMSA
E11D 09             SPSRS2 DEX
E11E 9C 55          CPX      BMWA
E120 27 24          BEQ      SPSRXF    EXIT IF BMS FULL
*

```

```

E122 DE 57          LDX      BMRA      DEC BMRA BY 1
E124 26 03          BNE      SPSRS3    WITH W.A.
E126 CE 07FD        LDX      #BMSA
E129 09             SPSRS3 DEX
E12A DF 57          STX      BMRA
E12C 0A             CLV              IND. PAGE SAVED
E12D 39             RTS
*

```

```

* CASE 3          * WRITE NOT ACTIVE AND NOT REQUESTED
*

```

```

E12E D6 59          SPSRC3 LDA B  BSS      (BSS UP TO DATE)
E130 2B 14          BMI      SPSRXF    EXIT IF BMS FULL
E132 DE 57          LDX      BMRA      DEC BMRA BY 1

```



```

E134 26 03          BNE      SPSRS5    WITH W.A.
E136 CE 07FD          LDX      #BMSA
E139 09              SPSRS5 DEX
E13A DF 57              STX      BMRA
E13C 9C 55              CPX      BMWA    UPDATE BSS
E13E 26 04          BNE      SPSRS6    SKIP IF BMS NOT FULL
E140 C6 0A          LDA B    #10      INDICATE BMS FULL
E142 D7 59          STA B    BSS
E144 0A              SPSRS6 CLV      IND. PAGE SAVED
E145 39              RTS
*
E146 0B              SPSRXF SEV      IND. PAGE NOT SAVED
E147 39              RTS
*
**  END  **
E148 000C          RMB      12
*
*
E154      FDID      EQU      *          FIND I. D.
*
* - SUBROUTINE TO FIND THE LOCATION OF THE I.D.
* - OF AN ENTRY IN A LOOKUP TABLE
* - UPON ENTRY:      X = LOCATION OF TABLE
*                   A = I.D. OF ENTRY IN TABLE
*                   B = BYTES/ENTRY IN TABLE
* - UPON EXIT :      X = LOCATION OF FOUND ENTRY I.D.
*                   CARRY BIT IS (CLEAR/SET) IF I.D.
*                   (IS/IS NOT) FOUND
* - CHANGES X,CCR
*
E154 37              PSH B          *** SAVE B,BYTES/ENTRY
E155 A1 00      FDIDL1 CMP A    0,X      CHECK IF I.D. FOUND
E157 27 0D              BEQ      IDFND    BRANCH IF FOUND
E159 E6 00              LDA B    0,X      CHECK FOR ENDOFTBL.
E15B 53              COM B          IE) $FF
E15C 27 0B              BEQ      FDIDX1    EXIT IF ENDOFTABLE
*
E15E 33              PUL B          *** RESTORE BYTES/ENTRY
E15F 37              PSH B          *** SAVE # BYTES/ENTRY
E160 08      FDIDL2 INX              INC. TO NEXT ENTRY
E161 5A              DEC B
E162 26 FC          BNE      FDIDL2
E164 20 EF          BRA      FDIDL1    TEST NEXT ENTRY
*
E166 33      IDFND PUL B          *** RESTORE STACK
E167 0C              CLC              IND. ENTRY FOUND
E168 39              RTS              RETURN
*
E169 33      FDIDX1 PUL B          *** RESTORE STACK
E16A 0D              SEC              IND. ENTRY NOT FOUND
E16B 39              RTS              RETURN
*
**  END  **
*

```



```

*
E16C      GTBT      EQU      *      GET BYTE
*
* - A SUBROUTINE TO GET A BYTE FORM THE DISPLAY
*   WHICH THE USER ENTERS FROM THE KEYBOARD
* - READS AND RETURNS THE FIRST AVAILABLE HEXADECIMAL
*   BYTE ENTERED IN THE SPECIFIED LOCATION
* - INPUTS : LEDNO = # OF LED WHERE THE BYTE IS TO
*             BE ENTERED (NOT TO BE ZERO)
*             THE DISPLAY PATTERN IN THAT AND THE
*             FOLLOWING LED
* - OUTPUTS: A = HEX BYTE ENTERED
*             C IN CCR SET IF BYTE ENTRY ABORTED AS
*             THE DISPLAY WAS CLEARED AND
*             CLEAR IF A BYTE PROPERLY ENTERED
* - CHANGES C IN CCR, A,B,X,LEDNO
* - ALLOWS INTERRUPTS WITHIN ROUTINE
*
E16C 07          TPA
E16D 36          PSH A          *** SAVE INT. MASK
E16E 96 12       LDA A LEDNO
E170 36          PSH A          ** SAVE LOC. OF BYTE
*
E171 0E          GTBTL1 CLI          ALLOW FOR INT.
E172 BD FFF1     JSR          DELAY2 WAIT .1 SEC.
E175 0030        FDB          $0030 FOR ENTRY
E177 0F          SEI          DO NOT ALLOW INT.
E178 96 12       LDA A LEDNO
E17A 27 0E       BEQ          GTBTX2 EXIT IF DISP. CLR.
E17C 32          PUL A          **
E17D 36          PSH A          **
E17E BD FFC1     JSR          READ2  TRY TO READ BYTE
E181 24 0C       BCC          GTBTX1 EXIT IF ENTERED
E183 96 12       LDA A LEDNO
E185 BD FFBB     JSR          WRITE0 WRITE TO NEXT LED
E188 20 E7       BRA          GTBTL1 DO UNTIL BYTE READ
*
E18A 32          GTBTX2 PUL A          ** REMOVE BYTE LOC.
E18B 32          PUL A          *** RESTORE INT. MASK
E18C 06          TAP
E18D 0D          SEC          IND. ENTRY ABORTED
E18E 39          RTS
*
E18F 16          GTBTX1 TAB          SAVE A
E190 32          PUL A          ** REMOVE BYTE LOC.
E191 32          PUL A          *** RESTORE INT. MASK
E192 06          TAP
E193 17          TBA          RESTORE A
E194 0C          CLC          IND. BYTE ENTERED
E195 39          RTS
*
**      END      **
*
*
```


E196 FLAG EQU * FLAG

*

- * - KEYBOARD ACTIVATED FUNCTION
- * - THE FIRST TWO CHARACTERS ENTERED INTO THE
DISPLAY WILL BE CHANGED INTO A HEX BYTE AND
STORED IN FLGC PROVIDED THE HEX BYTE READ
IS ALLOWED
- * - THE NEW VALUE OF FLGC WILL BE DISPLAYED IN THE
NEXT TWO CHARACTERS IN THE DISPLAY
- * - FLGC WILL BE ATTACHED TO THE END OF THE NEXT
RESAMPLED BLOCK AND TXED AND STORED ALONG WITH
THE SAMPLING DATA

*

```

E196 96 89          LDA A  FTNFLG
E198 27 07          BEQ    FLAGS0
E19A CE F423        LDX    #ERRORT   DISP. "ERROR",EXIT
E19D BD FFD6        JSR    DTSR2     IF OTHER FTN.ACTIVE
E1A0 3B             RTI

```

*

```

E1A1 96 11  FLAGS0 LDA A  SECFLG   IS SECFLG SET?
E1A3 27 06          BEQ    FLAGS1   SKIP IF NOT SET

```

*

```

E1A5 BD FBF6        JSR    SCNDSR
E1A8 7E E1D9        JMP    TIME     GO TO TIME IF SET

```

*

```

E1AB 96 42  FLAGS1 LDA A  FLGC     EXIT IF FLGC NOT CLR,
E1AD 26 11          BNE    FLAGX1   (NOT TXED YET)

```

*

```

E1AF BD FFC1        JSR    READ2    GET 1ST BYTE IN DISP.
E1B2 25 0C          BCS    FLAGX1   EXIT IF ERROR
E1B4 CE E1C6        LDX    #FLGTBL  GET FLAG TABLE
E1B7 C6 01          LDA B  #$01
E1B9 BD E154        JSR    FDID     IF BYTE FOUND
E1BC 24 02          BCC    FLAGX1   THEN EXIT

```

*

```

E1BE 97 42          STA A  FLGC     STORE VALID BYTE

```

*

```

E1C0 96 42  FLAGX1 LDA A  FLGC     GET FLGC FOR VARIFY
E1C2 BD FFC4        JSR    WRITE2   DISPLAY FLGC
E1C5 3B             RTI             EXIT

```

** END **

*

*

E1C6 FLGTBL EQU * FLAG TABLE

*

- * - TABLE OF FLAGS NOT ALLOWED
- * - ONE BYTE / ENTRY

*

```

E1C6 C1  FCB $C1,$C2,$C3,$C4,$C5,$C6,$C7,$C8  1ST,2ND
E1C7 C2
E1C8 C3
E1C9 C4
E1CA C5
E1CB C6

```



```

E1CC C7
E1CD C8
E1CE D1   FCB $D1,$D2,$D3,$D4,$D5,$D6,$D7,$D8   CAL.  FLAGS
E1CF D2
E1D0 D3
E1D1 D4
E1D2 D5
E1D3 D6
E1D4 D7
E1D5 D8
E1D6 F0   FCB  $F0,$00           FIRST SAMPLE,NO FLAG
E1D7 00
E1D8 FF   FCB  $FF           'END OF TABLE'

```

```

*
*
E1D9  TIME EQU  *   TIME
*
* - KEYBOARD ACTIVATED FUNCTION
* - USED TO SET THE TIME COUNTERS IN PTM #2
* - USES THE BCD DATA DISPLAYED IN THE FIRST
*   6 LED'S OF THE DISPLAY TO SET THE COUNTERS
* - INPUT : - THE FIRST PAIR OF LED'S ARE TO
*             DISPLAY HOURS, THE NEXT PAIR
*             MINUTES, AND THE NEXT PAIR
*             SECONDS
* - OUTPUT: - "TS" IS DISPLAYED IN THE LAST
*             PAIR OF LED'S IF THE TIMER WAS SET
*             - THE DISPLAY IS CLEARED IF THE
*             TIMER WAS NOT SET
* - THE TIMER IS NOT SET UNDER THE FOLLOWING
*   CONDITIONS:
*       - THE FIRST PAIR GREATER THAN "23"
*       - EITHER OF THE NEXT TWO PAIRS
*         GREATER THAN "58" OR IF THEY
*         CONTAIN A "9"
*       ( THE PTM WILL NOT OPERATE
*         PROPERLY IF SET TO 59 MINUTES
*         OR SECONDS )

```

```

E1D9 96 87          LDA A   MLAC
E1DB 27 07          BEQ     TIMES0
*
E1DD CE F423        LDX     #ERRORT   DISP."ERROR", EXIT
E1E0 BD FFD6        JSR     DTSR2     IF WE ARE TXING
E1E3 3B             RTI
*
E1E4 86 70   TIMES0 LDA A   #$70      READ DISP.LOC. 0
E1E6 B7 8001   STA A   PKDICR  WITH AUTOINCREMENT
E1E9 BD FFC7   JSR     RDWL     GET 1ST CHAR.
E1EC 02        FCB     $02      MUST BE .LE. 2
E1ED 97 04     STA A   ENTRY1
E1EF 81 02     CMP A   #$02      TEST 1ST CHAR.
E1F1 26 06     BNE     TIMES1    GET 2ND CHAR.

```

```

*
```


E1F3	BD	FFC7		JSR	RDWL	1ST CHAR. WAS 2
E1F6	03			FCB	\$03	NEXT MUST BE .LE. 3
E1F7	20	04		BRA	TIMES2	
*						
E1F9	BD	FFC7	TIMES1	JSR	RDWL	1ST CHAR. WAS 0 OR 1
E1FC	09			FCB	\$09	NEXT MUST BE .LE. 9
E1FD	97	05	TIMES2	STA A	ENTRY2	
E1FF	BD	FFC7		JSR	RDWL	
E202	05			FCB	\$05	
E203	97	06		STA A	ENTRY3	
E205	BD	FFC7		JSR	RDWL	
E208	08			FCB	\$08	
E209	97	07		STA A	ENTRY4	
E20B	BD	FFC7		JSR	RDWL	
E20E	05			FCB	\$05	
E20F	97	08		STA A	ENTRY5	
E211	BD	FFC7		JSR	RDWL	
E214	08			FCB	\$08	
E215	97	09		STA A	ENTRY6	
*						
E217	96	04		LDA A	ENTRY1	1ST CHAR. TO HEX
E219	81	02		CMP A	#\$02	
E21B	26	04		BNE	TIMES3	
*						
E21D	C6	14		LDA B	#\$14	B = 20
E21F	20	09		BRA	TIMES5	
*						
E221	81	01	TIMES3	CMP A	#\$01	
E223	26	04		BNE	TIMES4	
*						
E225	C6	0A		LDA B	#\$0A	B = 10
E227	20	01		BRA	TIMES5	
*						
E229	5F		TIMES4	CLR B		B = 0
E22A	DB	05	TIMES5	ADD B	ENTRY2	B = # OF HOURS
*						
E22C	86	84		LDA A	#\$84	ENABLE INIT.ON WRITE TO
E22E	B7	9400		STA A	CR4	TIMER LATCHES,Q4 IN M,L
E231	86	84		LDA A	#\$84	WRITE TO CR6,Q5 IN M,L
E233	B7	9401		STA A	CR5	
E236	86	80		LDA A	#\$80	NO 1/8 CLK,Q6 IN N MODE
E238	B7	9400		STA A	CR6	
*						
E23B	7F	9406		CLR	TL6H	STORE TIME IN LATCHES
E23E	86	17		LDA A	#\$17	
E240	10			SBA		
E241	B7	9407		STA A	TL6H+1	
E244	86	05		LDA A	#\$05	
E246	C6	09		LDA B	#\$09	
E248	90	06		SUB A	ENTRY3	
E24A	D0	07		SUB B	ENTRY4	
E24C	B7	9404		STA A	TL5H	
E24F	F7	9405		STA B	TL5H+1	
E252	86	05		LDA A	#\$05	


```

E254 C6 09          LDA B   #$09
E256 90 08          SUB A   ENTRY5
E258 D0 09          SUB B   ENTRY6
E25A B7 9402        STA A   TL4H
E25D F7 9403        STA B   TL4H+1
*
E260 86 90          LDA A   #$90  DISABLE CNTR INIT. ON
E262 B7 9400        STA A   CR6   WRITE TO TIMER LATCHES
E265 86 95          LDA A   #$95  WRITE TO CR4
E267 B7 9401        STA A   CR5
E26A 86 94          LDA A   #$94
E26C B7 9400        STA A   CR4
*
E26F CE 0017        LDX     #$0017  SET TIMER LATCHES
E272 FF 9406        STX     TL6H   FOR PROPER OVERFLOW
E275 CE 0509        LDX     #$0509
E278 FF 9404        STX     TL5H
E27B FF 9402        STX     TL4H
*
E27E 86 96          LDA A   #$96  WRITE TO LED #6
E280 B7 8001        STA A   PKDICR
E283 86 45          LDA A   #$45  DISPLAY "TS."
E285 C6 EE          LDA B   #$EE
E287 B7 8000        STA A   DISRAM
E28A F7 8000        STA B   DISRAM
E28D 86 90          LDA A   #$90  WRITE TO LED #0
E28F B7 8001        STA A   PKDICR
E292 3B             RTI          EXIT
*
**  END  **
*
*
*
*
*
E293 CAL EQU  *  CALIBRATE
*
* - A KEYBOARD FUNCTION TO GET A SAMPLE
* BLOCK WHERE THE INPUTS TO ONE OF THE
* CHANNELS IS A REFERENCE LEVEL
* - USED TO CALIBRATE THE DATA GATHERED
* - A ONE BYTE CODE MUST BE ENTERED TO
* SPECIFY THE CHANNEL NUMBER AND WHETHER
* A ZERO OR HIGH LEVEL REFERENCE INPUT
* IS BEING USED
* - $C1 TO $C8 INDICATE A ZERO INPUT REFERENCE
* LEVEL ON ONE OF CHANNELS 1 TO 8
* - $D1 TO $D8 INDICATE A HIGH INPUT REFERENCE
* LEVEL
* - TAKES A NUMBER OF SAMPLES AND FILTERS THEM
* SO AN AVERAGE IS OBTAINED AND THE D.C.
* GAIN OF THE FILTER SELECTED IS ACCOUNTED
* FOR
*
*
*

```



```

E293 96 11      LDA A  SECFLG  CHECK IF 2ND FTN.
E295 27 06      BEQ    CALS1    SELECTED
*
E297 BD FBF6    JSR      SCNDSR
E29A 7E BDFD    JMP      SIG      GO TO 2ND FUNCTION
*
E29D 96 89      CALS1  LDA A  FTNFLG
E29F 26 36      BNE      CALXE    EXIT IF FTN> ACTIVE
E2A1 96 86      LDA A  SYST
E2A3 85 F0      BIT A  #$F0      CONTINUE IF
E2A5 27 07      BEQ      CALS2    NOT SAMPLING
*
E2A7 CE BF58    LDX      #DUSEHS  DISP."USE .HS" TO
E2AA BD FFD6    JSR      DTSR2    PROMPT USER TO STOP
E2AD 3B          RTI              SAMPLING
*
E2AE 96 8B      CALS2  LDA A  SELFLG  CONTINUE IF FILTER
E2B0 26 07      BNE      CALS3    WAS SELECTED
*
E2B2 CE BF28    LDX      #DUSEL    DISP."USE SEL" AS
E2B5 BD FFD6    JSR      DTSR2    FILTER NOT SELECTED
E2B8 3B          RTI
*
E2B9 7C 0089    CALS3  INC      FTNFLG  IND. A FTN. ACTIVE
E2BC BD FFD0    JSR      CLDISP
E2BF CE E380    LDX      #DCALT    DISPLAY "CAL." TO
E2C2 86 03      LDA A  #$03      PROMPT USER TO ENTER
E2C4 5F          CLR B              CALIBRATION CODE
E2C5 BD E5D8    JSR      DSCT
*
E2C8 BD E16C    JSR      GTBT      GET ONE BYTE CODE
E2CB 25 10      BCS      CALX1    EXIT IF DISP.CLRED.
E2CD CE E383    LDX      #CALTBL  GET CODE TABLE
E2D0 C6 01      LDA B  #$01      WITH 1 BYTE / ENTRY
E2D2 BD E154    JSR      FDID      FIND THE CODE
E2D5 24 0A      BCC      CALS4    CONTINUE IF FOUND
*
E2D7 CE F423    CALXE  LDX      #ERRORT  DISPLAY "ERROR"
E2DA BD FFD6    JSR      DTSR2
E2DD 7F 0089    CALX1  CLR      FTNFLG  IND.FTN. NOT ACTIVE
E2E0 3B          RTI
*
E2E1 97 42      CALS4  STA A  FLGC      SAVE CODE AS FLAG
E2E3 96 85      LDA A  SPLCRS  GET SAMPLE CNTR RESET
E2E5 CE E394    LDX      #CSNTBL  GET SAMPLE # TABLE
E2E8 C6 03      LDA B  #$03      3 BYTES / ENTRY
E2EA BD E154    JSR      FDID      FIND LOC.OF RESET
E2ED 25 E8      BCS      CALXE    EXIT IF NOT FOUND
E2EF EE 01      LDX      1,X      SAVE # OF SAMPLES
E2F1 DF 0A      STX      ENTRY7   TO BE TAKEN
E2F3 DE 32      LDX      SPLVEC   SAVE VECTOR TO
E2F5 DF 08      STX      ENTRY5   SAMPLE PROCESS
E2F7 CE E721    LDX      #VCALP   ACTIVATE CAL.
E2FA DF 32      STX      SPLVEC   PROCESS

```



```

E2FC CE 0400      LDX      #S      CLEAR FILTER DATA
E2FF 6F 00      CALL 1  CLR      0,X
E301 08          INX
E302 8C 04A6     CPX      #Y+22
E305 26 F8      BNE      CALL 1
E307 86 D7      LDA  A   #$D7      ENABLE TIMER INT.
E309 B7 7C01     STA  A   CR2      ( CLK #2 )
E30C 3B          RTI
*
E30D CALP EQU *  CALIBRATION PROCESS
*
* - ENTERED UPON TIMER INTERRUPT ( CLK #2 )
* - GETS A NUMBER OF SAMPLES AND FILTERS
*   THEM
* - AFTER THE LAST SAMPLE THE CALIBRATION
*   CODE IS ADDED TO THE SAMPLE BLOCK AS
*   A FLAG CHARACTER AND THE LINE OF DATA
*   IS STORED IN THE BMS
*
*
E30D 0E          CLI          ALLOW FOR INT.
E30E BD F3D6     JSR      SAMP   OBTAIN SAMPLE BLK.
E311 01          NOP
*
E312 BD EFD0     JSR      FILT   FILTER,S IN,Y OUT
*
E315 DE 0A      LDX      ENTRY7  DEC. SAMPLE COUNT
E317 09          DEX
E318 DF 0A      STX      ENTRY7  CONTINUE IF THIS IS
E31A 27 01      BEQ      CALPS1  THE LAST SAMPLE
*
E31C 3B          RTI
*
E31D 86 97      CALPS1 LDA  A   #$97      DISABLE CLK2,PTM1
E31F B7 7C01     STA  A   CR2      INTERRUPT
E322 DE 08      LDX      ENTRY5  RESTORE VECTOR TO
E324 DF 32      STX      SPLVEC  SAMPLE PROCESS
E326 CE 0490     LDX      #Y      MOVE OUTPUT TO BSIL
E329 BD F3F4     JSR      MDTI    ( 16 BYTES )
E32C BD F3EB     JSR      CKSM    APPEND CHECK SUM(2)
E32F BD F3E5     JSR      TMHB    ADD TIME (3 BYTES)
E332 96 42      LDA  A   FLGC
E334 B7 0666     STA  A   BSIL+22  ADD 1 FLAG BYTE
E337 7F 0042     CLR      FLGC
*
E33A 86 80      LDA  A   #FRFR  FRAME THE SAMP.BLOCK
E33C B7 0650     STA  A   BSIL
E33F 86 81      LDA  A   #FREN
E341 B7 0667     STA  A   BSIL+23
*
E344 C6 18      CALPL1 LDA  B   #24      # OF BYTES IN BSIL
E346 CE 0650     LDX      #BSIL
E349 BD B7E6     JSR      BMSI    STORE LINE IN BMS
E34C 28 0A      BVC      CALPS2  SKIP IF BMS NOT FULL

```



```

*
E34E CE BF20      LDX      #DBMSFL   DISPLAY "BMS FULL"
E351 BD FFD6      JSR      DTSR2
E354 7F 0089      CLR      FTNFLG   IND.FTN. INACTIVE
E357 3B           RTI
*
E358 27 EA      CALPS2 BEQ      CALPL1   RETRY IF NOT STORED
*
E35A BD FFD0      JSR      CLDISP
E35D CE E380      LDX      #DCALT   DISPLAY "CAL."
E360 86 03        LDA      A      #$03
E362 5F          CLR      B
E363 BD E5D8      JSR      DSCT
E366 CE E740      LDX      #D104TB   DISPLAY "10 4"
E369 86 04        LDA      A      #$04
E36B C6 04        LDA      B      #$04
E36D BD E5D8      JSR      DSCT
E370 7F 0089      CLR      FTNFLG   IND. FTN. INACTIVE
E373 3B           RTI
*
**      END      **
E374 000C          RMB      12
*
*
E380 DCALT      EQU *      DISPLAY CALIBRATE
*
E380 78          FCB      $78      "C"
E381 77          FCB      $77      "A"
E382 B8          FCB      $B8      "L."
*
*
E383 CALTBL EQU *      CALIBRATION TABLE
*
*      - FORMAT :
*          FCB ( CALIBRATION CODE )
*
*      ZERO REFERENCE LEVELS, CANNELS 1 TO 8
*
E383 C1      FCB      $C1,$C2,$C3,$C4,$C5,$C6,$C7,$C8
E384 C2
E385 C3
E386 C4
E387 C5
E388 C6
E389 C7
E38A C8
*
*      HIGH REFERENCE LEVELS, CANNELS 1 TO 8
*
E38B D1      FCB      $D1,$D2,$D3,$D4,$D5,$D6,$D7,$D8
E38C D2
E38D D3
E38E D4
E38F D5

```



```

E390 D6
E391 D7
E392 D8
E393 FF      FCB      $FF      " END OF TABLE "
*
*
E394 CSNTBL EQU * CALIBRATION SAMPLE NUMBER TABLE
*
*   - FORMAT:
*       FCB ( SAMPLE COUNTER RESET VALUE )
*       FDB ( # OF SAMPLES FILTERED FOR
*             CALIBRATION )
*
E394 14      FCB      20      2 HZ. RESAMPLE
E395 0100    FDB      256
*
E397 0A      FCB      10      4 HZ. RESAMPLE
E398 00FA    FDB      250
*
E39A 05      FCB      5       8 HZ. RESAMPLE
E39B 0087    FDB      135
*
E39D 04      FCB      4       10 HZ. RESAMPLE
E39E 008F    FDB      $8F
*
E3A0 FF      FCB      $FF      " END OF TABLE "
*
*
*
E3A1 BDB EQU * BUBBLE MEMORY DEBUG ROUTINES
*   -----
*
*   - ENTRY POINT TO A SET OF ROUTINES TO CHECK
*     THE BUBBLE MEMORY SYSTEM
*   - TO EXECUTE ONE OF THE PROGRAMS ENTER:
*     * , .BDB , X , X
*     WERE XX IS THE I.D. OF THE DESIRED ROUTINE
*
E3A1 BD FFD0      JSR      CLDISP      CLR DISPLAY, LEDNO
E3A4 86 3E        LDA A   #$3E        DISPLAY "B"
E3A6 B7 8000      STA A   DISRAM
E3A9 7C 0012      INC      LEDNO
*
E3AC 0E          BDBL1  CLI          ALLOW INTERRUPTS
E3AD BD FFF1      JSR      DELAY2     DELAY .1 SEC
E3B0 0030        FDB      $0030
E3B2 0F          SEI          DO NOT ALLOW INT.
E3B3 96 12        LDA A   LEDNO      DISPLAY CLEARED?
E3B5 27 29        BEQ      BDBX1     EXIT IF SO
*
E3B7 86 71        LDA A   #$71       READ BYTE1 OF DISRAM
E3B9 B7 8001      STA A   PKDICR     WITH AUTO INCREMENT
E3BC B6 8000      LDA A   DISRAM
E3BF F6 8000      LDA B   DISRAM     GET NEXT TWO BYTES

```



```

E3C2 BD FFE2      JSR      DSTOBI      CONVERT TO HEX
E3C5 24 09        BCC      BDBS1      SKIP IF ROUTINE #
E3C7 86 90        LDA A    #$90       WRITE TO DISRAM
E3C9 9B 12        ADD A    LEDNO      ( PROPER BYTE )
E3CB B7 8001      STA A    PKDICR
E3CE 20 DC        BRA      BDBL1      WAIT FOR ENTRY
*
E3D0 CE E4EE BDBS1 LDX      #BDBTBL   TABLE OF BDB ROUT.
E3D3 C6 03        LDA B    #$03      3 BYTES / ENTRY
E3D5 BD E154      JSR      FDID       FIND LOC. OF #
E3D8 24 02        BCC      BDBFND     BRANCH IF FOUND
E3DA 20 C5        BRA      BDB       RESTART,# NOT FOUND
*
E3DC EE 01      BDBFND LDX      1,X     GET BDB ROUTINE LOC.
E3DE AD 00      JSR      0,X     JUMP TO ROUTINE
*
E3E0 3B          BDBX1  RTI          EXIT
*
*
E3E1 RBCS EQU * READ BUBBLE CONTROLER STATUS
*
*   - READS AND DISPLAYS THE STATUS REGISTER
*   OF THE BUBBLE MEMORY CONTROLLER
*
E3E1 86 94      LDA A    #$94       WRITE TO
E3E3 B7 8001      STA A    PKDICR    DISRAM BYTE 4
E3E6 B6 AC01      LDA A    BCSR      BUB.CNTRL.STAT.REG.
E3E9 BD FFC4      JSR      WRITE2    DISPLAY STATUS
E3EC C6 06      LDA B    #$06       INDICATE READY TO
E3EE D7 12      STA B    LEDNO      WRITE TO BYTE 6
E3F0 39          RTS                EXIT
**   END   **
*
*
E3F1 WRX1 EQU * WRITE REGISTERS VERSION X1
*
* - WRITES 7220 REGISTERS B THROUGH F WITH
*   VALUES SUITABLE TO DEBUG THE BM SYSTEM
* - ONLY HALF A BUBBLE IS SELECTED SO THAT
*   DATA MAY BE TRANSFERED WITHOUT PROGRAM
*   TIMING ERRORS
* - X1 :  B = $01 - 1 PAGE TRANSFERS
*         C = $00 - SELECT 1FSA CHANNEL, 1/2 BUB.
*         D = $08 - LOW FREQ., 50KHZ FIELD 4MHZ CLK
*         E = $00 - PAGE 0, START ADDRESS $000
*         F = $00 - SELECT FIRST BUBBLE
E3F1 86 0B      LDA A    #$0B       READY TO WRITE TO B
E3F3 B7 AC01      STA A    BCSR      (WITH AUTO INCREMENT)
E3F6 86 01      LDA A    #$01
E3F8 B7 AC00      STA A    BCDR
E3FB 86 00      LDA A    #$00
E3FD B7 AC00      STA A    BCDR
E400 86 08      LDA A    #$08
E402 B7 AC00      STA A    BCDR

```



```

E405 86 00          LDA A   #$00
E407 B7 AC00        STA A   BCDR
E40A 86 00          LDA A   #$00
E40C B7 AC00        STA A   BCDR
E40F 20 53          BRA     BDBCX1   GO TO COMMON EXIT 1

```

*

*

```

E411 WRX2 EQU *   WRITE REGISTERS VERSION   X2

```

*

* - SAME AS WRX1 EXCEPT A WHOLE BUBBLE, (2 FSA CH.),
* IS SELECTED

* - USED TO DEBUG WRITE/READ BOOTLOOP REGISTERS,
* CHECK FOR MISSING SEEDS, ECT.

* - X2 : B = \$01 - 1 PAGE TRANSFERS

* C = \$10 - 2 FSA CHANNELS,WHOLE BUBBLE

* D = \$08 - LOW FREQ.,50KHZ FIELD 4MHZ CLK.

* E = \$00 - PAGE 0,START ADD.= \$000

* F = \$00 - SELECT FIRST BUBBLE

*

```

E411 86 0B          LDA A   #$0B   READY TO WRITE TO B
E413 B7 AC01        STA A   BCSR   (WITH AUTOINCREMENT)
E416 86 01          LDA A   #$01
E418 B7 AC00        STA A   BCDR
E41B 86 10          LDA A   #$10
E41D B7 AC00        STA A   BCDR
E420 86 08          LDA A   #$08
E422 B7 AC00        STA A   BCDR
E425 86 00          LDA A   #$00
E427 B7 AC00        STA A   BCDR
E42A 86 00          LDA A   #$00
E42C B7 AC00        STA A   BCDR
E42F 20 33          BRA     BDBCX1   GO TO COMMON EXIT 1

```

*

*

```

E431 WRFF EQU *   WRITE THE FIFO

```

*

* - WRITES 40 BYTES FROM FIFOI(RAM) TO THE 7220 FIFO

*

```

E431 CE 0600        LDX     #FIFOI   LOC. OF FIFO INPUT
E434 A6 00          WRFFL1 LDA A   0,X   GET BYTE
E436 B7 AC00        STA A   BCDR   WRITE TO DATA REG.
E439 08             INX             GO TO NEXT BYTE
E43A 8C 0628        CPX     #FIFOI+40 ALL BYTES WRITTEN?
E43D 26 F5          BNE     WRFFL1   LOOP IF NOT
E43F 20 23          BRA     BDBCX1   GO TO COMMON EXIT 1

```

*

*

```

E441      RDFF      EQU      *          READ FIFO

```

*

* - READS 40 BYTES FROM BMC FIFO AND SOTRES
* THEM IN FIFO0

*

```

E441 CE 0628        LDX     #FIFO0   LOC. OF FIFO OUTPUT
E444 B6 AC00 RDFFL1 LDA A   BCDR   BYTE FROM DATA REG.

```



```

E447 A7 00          STA A 0,X          STORE BYTE IN FIFO0
E449 08             INX
E44A 8C 0650        CPX      #FIFO0+40 IF ALL BYTES READ?
E44D 26 F5          BNE      RDFFL1     LOOP IF NOT
E44F 20 13          BRA      BDBCX1     GO TO COMMON EXIT 1

```

*

*

```

E451      GFFD      EQU      *          GENERATE FIFO DATA

```

*

* - ENTERS A TEST PATTERN OF 40 BYTES INTO FIFOI

*

```

E451 CE 0600        LDX      #FIFOI     LOC. OF FIFO INPUT
E454 4F             CLR A
E455 C6 AA          LDA B $AA          GENERATE A
E457 A7 00          GFFDL1 STA A 0,X     SERIES OF PAIRS
E459 4C             INC A              FIRST OF EACH PAIR
E45A 08             INX                COUNTS
E45B E7 00          STA B 0,X          SECOND OF EACH PAIR
E45D 53             COM B              ALTERNATES BETWEEN
E45E 08             INX                $AA AND $55
E45F 8C 0628        CPX      #FIFOI+40 ALL GENERATED?
E462 26 F3          BNE      GFFDL1     LOOP IF NOT
E464 BD FFD0        BDBCX1 JSR      CLDISP COMMON BDB EXIT 1
E467 39             RTS

```

*

*

```

E468 VFFD EQU *    VERIFY FIFO DATA

```

*

* - COMPARES CONSECUTIVE BYTES OF FIFOI WITH
 * CORRESPONDING BYTES IN FIFO0
 * - IF THE TWO BUFFERS ARE THE SAME, "10 4" IS DISP.
 * AND WE ARE READY TO WRITE TO LED #0
 * - IF A PAIR OF BYTES DO NOT AGREE, THE LOC. IN
 * FIFOI IS DISPLAYED TOGETHER WITH THE PAIR
 * (FIFOI,X),(FIFO0,X)
 * - TO EXIT WHEN THERE ARE ERRORS, CLEAR DISPLAY

*

```

E468 CE 0600        LDX      #FIFOI     FIFOI,FIFO0 ADJACENT
E46B A6 00          VFFDL1 LDA A 0,X     GET INPUT BYTE
E46D A1 28          CMP A 40,X          COMPARE WITH OUTPUT
E46F 26 15          BNE      VFFDE1     BRANCH IF NOT SAME
E471 08            VFFDR1 INX
E472 8C 0628        CPX      #FIFOI+40 ALL PAIRS TESTED?
E475 26 F4          BNE      VFFDL1     LOOP IF NOT

```

*

```

E477 CE E740        VFFDX2 LDX      #D104TB GET "10 4" TABLE
E47A BD FFD6        JSR      DTSR2     DISPLAY "10 4"
E47D 86 90          VFFDX1 LDA A $90    READY TO WRITE LED0
E47F B7 8001        STA A PKDICR
E482 7F 0012        CLR      LEDNO
E485 39             RTS              EXIT

```

*

```

E486 DF 0A          VFFDE1 STX      ENTRY7 SAVE X
E488 E6 28          LDA B 40,X        GET OUTPUT BYTE

```



```

E48A BD FFDF      JSR      DSPXAB      CLDISP.,DISP. X,A,B
*
E48D 0E          VFFDL2 CLI          ALLOW INTERRUPTS
E48E BD FFF1      JSR      DELAY2
E491 0030          FDB      $0030      DELAY FOR .1 SEC.
E493 0F          SEI          DO NOT ALLOW INT.
E494 86 60          LDA A    #$60      DISRAM READ
E496 B7 8001          STA A    PKDICR
E499 B6 8000          LDA A    DISRAM      READ BYTE 0
E49C 27 DF          BEQ      VFFDX1      GO TO EXIT IF CLEAR
*
E49E 96 11          LDA A    SECFLG      2ND FTN. KEY?
E4A0 27 EB          BEQ      VFFDL2      LOOP FOREVER IF NOT
*
E4A2 BD FFDO      JSR      CLDISP
E4A5 DE 0A          LDX      ENTRY7      RESTORE X
E4A7 20 C8          BRA      VFFDR1      RETURN TO LOOP 1
*
*
E4A9 MBLRW EQU * MASK BOOT LOOP REGISTER WRITE
*
E4A9 86 10          LDA A    #$10
E4AB 20 3A          BRA      BDBCX2
*
*
      E4AD      INIT      EQU      *          INITIALIZE
*
E4AD 86 11          LDA A    #$11
E4AF 20 36          BRA      BDBCX2
*
*
      E4B1      READ      EQU      *          READ
*
E4B1 86 12          LDA A    #$12
E4B3 20 32          BRA      BDBCX2
*
*
      E4B5      WRITE     EQU      *          WRITE
*
E4B5 86 13          LDA A    #$13
E4B7 20 2E          BRA      BDBCX2
*
*
      E4B9      RDSK      EQU      *          READ SEEK
*
E4B9 86 14          LDA A    #$14
E4BB 20 2A          BRA      BDBCX2
*
*
E4BD WBLR EQU *          WRITE BOOT LOOP REGISTER
*
E4BD 86 16          LDA A    #$16
E4BF 20 26          BRA      BDBCX2
*

```



```

*
E4C1 WRBL EQU * WRITE BOOT LOOP
*
E4C1 86 17 LDA A #$17
E4C3 20 22 BRA BDBCX2
*
*
E4C5 RFSAS EQU * READ FSA STATUS
*
E4C5 86 18 LDA A #$18
E4C7 20 1E BRA BDBCX2
*
*
E4C9 ABORT EQU * ABORT
*
E4C9 86 19 LDA A #$19
E4CB 20 1A BRA BDBCX2
*
*
E4CD WRSK EQU * WRITE SEEK
*
E4CD 86 1A LDA A #$1A
E4CF 20 16 BRA BDBCX2
*
*
E4D1 RDBL EQU * READ BOOT LOOP
*
E4D1 86 1B LDA A #$1B
E4D3 20 12 BRA BDBCX2
*
*
E4D5 RDCD EQU * READ CORRECTED DATA
*
E4D5 86 1C LDA A #$1C
E4D7 20 0E BRA BDBCX2
*
*
E4D9 FFRST EQU * FIFO RESET
*
E4D9 86 1D LDA A #$1D
E4DB 20 0A BRA BDBCX2
*
*
E4DD PURGE EQU * PURGE
*
E4DD 86 1E LDA A #$1E
E4DF 20 06 BRA BDBCX2
*
*
E4E1 SWRST EQU * SOFTWARE RESET
*
E4E1 86 1F LDA A #$1F
E4E3 20 02 BRA BDBCX2
*

```



```

*
E4E5      RBLR      EQU *      READ BOOT LOOP REGISTER
*
E4E5 86 15      LDA A  #$15
E4E7 B7 AC01 BDBCX2 STA A  BCSR      WRITE COMMAND TO BMC
E4EA BD FFD0      JSR      CLDISP
E4ED 39      RTS      EXIT
*
**      END      **
*
*
E4EE      BDBTBL EQU *      BUBBLE DEBUG TABLE
*
*      - TABLE OF BDB ROUTINE NUMBERS, LOCATIONS
*      AND NAMES
*
E4EE 00      FCB      $00
E4EF E3E1      FDB      RBCS      READ BMC STATUS
E4F1 01      FCB      $01
E4F2 E3F1      FDB      WRX1      WRITE REG.VERSION X1
E4F4 02      FCB      $02
E4F5 E411      FDB      WRX2      WRITE REG.VERSION X2
E4F7 03      FCB      $03
E4F8 E431      FDB      WRFF      WRITE FIFO
E4FA 04      FCB      $04
E4FB E441      FDB      RDFF      READ FIFO
E4FD 05      FCB      $05
E4FE E451      FDB      GFFD      GENERATE FIFO DATA
E500 06      FCB      $06
E501 E468      FDB      VFFD      VERIFY FIFO DATA
E503 07      FCB      $07      MASK BOOT
E504 E4A9      FDB      MBLRW      LOOP REG.WRITE
E506 08      FCB      $08
E507 E4AD      FDB      INIT      INITIALIZE
E509 09      FCB      $09
E50A E4B1      FDB      READ      READ
E50C 0A      FCB      $0A
E50D E4B5      FDB      WRITE      WRITE
E50F 0B      FCB      $0B
E510 E4B9      FDB      RDSK      READ SEEK
E512 0C      FCB      $0C
E513 E4BD      FDB      WBLR      WRITE BOOT LOOP REG.
E515 0D      FCB      $0D
E516 E4C1      FDB      WRBL      WRITE BOOT LOOP
E518 0E      FCB      $0E
E519 E4C5      FDB      RFSAS      READ FSA STATUS
E51B 00      FCB      $00      ABORT (NOT TO BE
E51C E4C9      FDB      ABORT      USED WITH 7220BRD)
E51E 10      FCB      $10
E51F E4CD      FDB      WRSK      WRITE SEEK
E521 11      FCB      $11
E522 E4D1      FDB      RDBL      READ BOOT LOOP
E524 12      FCB      $12
E525 E4D5      FDB      RDCD      READ CORRECTED DATA

```



```

E527 13          FCB      $13
E528 E4D9        FDB      FFRST  FIFO RESET
E52A 14          FCB      $14
E52B E4DD        FDB      PURGE  PURGE
E52D 15          FCB      $15
E52E E4E1        FDB      SWRST  SOFTWARE RESET
E530 16          FCB      $16
E531 E4E5        FDB      RBLR   READ BOOT LOOP REG.
E533 20          FCB      $20
E534 E546        FDB      BCXT   COMMAND EXECUTION TIME
E536 23          FCB      $23
E537 E53A        FDB      WRX3DB  WRITE REG.VER.3
E539 FF          FCB      $FF     "END OF TABLE"

```

*

*

```

E53A  WRX3DB EQU  *    WRX3  WITH BUBBLE DEBUG

```

*

```

* - ALLOWS WRX3 TO BE EXECUTED AS A BDB COMMAND
* - BUBBLE MEMORY CHIP #1 PAGE TO BE ACCESSED IS
*   PLACED IN ENTRY1 AS AN INPUT

```

*

```

E53A 96 04      LDA  A  ENTRY1
E53C D6 05      LDA  B  ENTRY2
E53E 37         PSH  B          ***  PUSH ADDRESS LSB
E53F 36         PSH  A          **   PUSH ADDRESS MSB
E540 BD E5F0    JSR      WRX3
E543 31         INS          **   REMOVE ADDRESS
E544 31         INS          ***   FORM STACK
E545 39         RTS

```

*

```

**  END  **

```

*

*

```

E546 BCXT EQU *    BUBBLE COMMAND EXECUTION TIME

```

*

```

* - DISPLAYS THE TIME REQUIRED TO EXECUTE CERTAIN
*   BMS ( BUBBLE MEMORY SYSTEM ) COMMANDS, NAMLY
*   THOSE LISTED IN BCXTT
* - UPON ENTRY, "BT" IS DISPLAYED, WHERE AFTER THE
*   BDB COMMAND CODE IS TO BE ENTERED
* - AFTER THE COMMAND IS SENT TO BMC AND AFTER
*   BMC IS NO LONGER BUSY, ELAPSED TIME IS DISPLAYED

```

*

```

E546 BD FFD0    JSR      CLDISP
E549 5F         CLR  B          STARTING WITH 1ST,
E54A 86 02      LDA  A  #$02    WRITE 2 LED'S
E54C CE E5C7    LDX      #DBTTBL WITH  "BT"
E54F BD E5D8    JSR      DSCT

```

*

```

E552 BD E16C    JSR      GTBT    GET COMMAND CODE
E555 24 03      BCC      BCXTS1  SKIP IF ENTERED

```

*

```

E557 7E E5C2    JMP      BCXTW   GO TO WAIT LOOP

```

*


```

E55A 7C 0000 BCXTS1 INC READEN ALLOW KEYBRD ENTRIES
E55D CE E5C9 LDX #BCXTT TIMABLE COMMANDS
E560 C6 01 LDA B #$01 1 BYTE / ENTRY
E562 BD E154 JSR FDID FIND CODE IN TABLE
E565 25 DF BCS BCXT RESTART IF NOT FOUND
*
E567 CE E4EE LDX #BDBTBL BDB COMMANDS
E56A C6 03 LDA B #$03
E56C BD E154 JSR FDID FIND CODE IN TABLE
E56F EE 01 LDX 1,X GET LOC.COMM. ROUTINE
E571 A6 01 LDA A 1,X GET COMMAND
*
E573 5F CLR B INIT.COUNTER BYTES
E574 D7 0B STA B ENTRY8
E576 D7 0A STA B ENTRY7
E578 D7 09 STA B ENTRY6
E57A B7 AC01 STA A BCSR START EXECUTION
*
E57D 96 0B BCXTL2 LDA A ENTRY8 3 INC. COUNTER
E57F 8B 51 ADD A #$51 2
E581 19 DAA 2 USE BCD ARITH.
E582 97 0B STA A ENTRY8 4
E584 96 0A LDA A ENTRY7 3
E586 89 00 ADC A #$00 2
E588 19 DAA 2
E589 97 0A STA A ENTRY7 4
E58B 96 09 LDA A ENTRY6 3
E58D 89 00 ADC A #$00 2
E58F 19 DAA 2
E590 29 0B BVS BCXTT0 4 EXIT ON OVERFLOW
E592 97 09 STA A ENTRY6 4
E594 F6 AC01 LDA B BCSR 4
E597 C5 80 BIT B #$80 2 SKIP IF EXECUTION
E599 27 0D BEQ BCXTS2 4 COMPLETE
E59B 20 E0 BRA BCXTL2 4/51 DO UNTIL IT IS
*
E59D 5F BCXTT0 CLR B STARTING AT 1ST LED
E59E 86 05 LDA A #$05 DISPLAY 5 CHARACTERS
E5A0 CE E5D3 LDX #DFOFT TO IND. OVERFLOW
E5A3 BD E5D8 JSR DSCT DISPLAY
E5A6 20 1A BRA BCXTW GO TO WAIT LOOP
*
E5A8 BD FFD0 BCXTS2 JSR CLDISP
E5AB 96 09 LDA A ENTRY6 DISPLAY EXECUTION
E5AD D6 0A LDA B ENTRY7 TIME
E5AF BD FFEB JSR WRITE4
E5B2 96 0B LDA A ENTRY8
E5B4 BD FFC4 JSR WRITE2
E5B7 86 05 LDA A #$05 AFTER LAST DIGIT
E5B9 BD FFCD JSR DSPDP PLACE DECIMAL POINT
E5BC B6 AC01 LDA A BCSR DISPLAY BMC STATUS
E5BF BD FFC4 JSR WRITE2
*
E5C2 4F BCXTW CLR A PREPARE TO WRITE

```



```

E5C3 BD FFBB      JSR      WRITE0    TO 1ST LED
E5C6 39           RTS      EXIT
*
*
      E5C7      DBTTBL EQU      *      DISPLAY "BT" TABLE
E5C7 3E           FCB      $3E      "B"
E5C8 45           FCB      $45      "T"
*
*
E5C9 BCXTT EQU *   TABLE OF ALLOWABLE BCXT COMMANDS
E5C9 0B           FCB      $0B      READ SEEK
E5CA 10           FCB      $10      WRITE SEEK
E5CB 13           FCB      $13      FIFO RESET
E5CC 15           FCB      $15      SOFTWARE RESET
E5CD 0E           FCB      $0E      READ FSA STATUS
E5CE 14           FCB      $14      PURGE
E5CF 08           FCB      $08      INITIALIZE
E5D0 FF           FCB      $FF,$FF,$FF "END OF TABLE"
E5D1 FF
E5D2 FF
*
*
E5D3 DTOFT EQU *   DISPLAY TIMER OVER FLOW TABLE
E5D3 C5           FCB      $C5      "T."
E5D4 00           FCB      0
E5D5 FD           FCB      $FD      "O."
E5D6 00           FCB      0
E5D7 F2           FCB      $F2      "F."
*
*
**      END      **
*
*
E5D8 DSCT EQU *    DISPLAY CHARACTER TABLE
*
* - ROUTINE TO DISPLAY A TABLE OF CHARACTERS
*   IN THE LED'S
* - INPUTS :  X = LOCATION OF TABLE
*              A = NO OF CHARACTERS TO BE DISPLAYED
*              B = NO. OF FIRST LED TO BE WRITTEN TO
*                ( FIRST LED = 0 )
* - OUTPUTS:  B = NO. OF NEXT LED TO BE WRITTEN
* - CHANGES  X,A,B,CCR,LEDNO
*
E5D8 C4 0F        AND B   #$0F      WRITE WITH
E5DA CA 90        ORA B   #$90      AUTOINCREMENT TO
E5DC F7 8001      STA B   PKDICR    LOC. SPECIFIED BY B
E5DF C4 0F        AND B   #$0F      REMOVE INSTRUCTION
*
E5E1 36          DSCTL1 PSH A      *** # CHAR. TO BE WRITTEN
E5E2 A6 00        LDA A   0,X
E5E4 B7 8000      STA A   DISRAM    DISPLAY CHAR.
E5E7 5C          INC B          NEXT LED
E5E8 08          INX          NEXT TABLE LOCATION

```



```

E5E9 32          PUL A      ***
E5EA 4A          DEC A
E5EB 26 F4       BNE        DSCTL1  DO UNTIL NONE LEFT
*
E5ED D7 12       STA B      LEDNO    NEXT LED WRITTEN
E5EF 39          RTS
*
**  END  **
*
*
E5F0 WRX3 EQU   *      WRITE REGISTERS VERSION 3
*
* - A ROUTINE TO WRITE BMC REGISTERS IN PREPERATION
*   FOR A COMMAND
* - REGISTER SET UP:(SEE BUBBLE SYSTEM DOCUMENTATION)
*   BLR=$1001,2 FSA CH., 1 PAGE/BUBBLE DATA TRANSFER
*   ER =$41  ,NO PARITY INT., ERROR INTERRUPTS ONLY ON
*               TIMING AND UNCORR. ERR., COMMAND INT.,
*               BOOTLOOP MASKED,MFBTR LOW,NO DMA
*   AR =%00000SSS,$SS , SELECT BUBBLE CHIP #1 ONLY,
*               S,S SELECTES DESIRED PAGE
*   RAC=$00  ,POINTS TO FIFI ON EXIT
* - INPUTS:- ADDRESS OF PAGE TO BE ACCESSED(TWO BYTES)
*               IS TO BE PUSHED ON STACK BEFORE ENTERY
* - NO OUTPUTS
* - CHANGES A,X,CCR
*
E5F0 86 0B       LDA A      #$0B
E5F2 B7 AC01     STA A      BCSR
E5F5 86 01       LDA A      #$01      WRITE BLR LSB
E5F7 B7 AC00     STA A      BCDR
E5FA 86 10       LDA A      #$10      WRITE BLR MSB
E5FC B7 AC00     STA A      BCDR
E5FF 86 41       LDA A      #$41      WRITE ER
E601 B7 AC00     STA A      BCDR
E604 30          TSX
E605 A6 03       LDA A      3,X      GET ADDRESS LSB
E607 B7 AC00     STA A      BCDR      WRITE ADDRESS LSB
E60A A6 02       LDA A      2,X      GET ADD. MSB
E60C 84 07       AND A      #$07      SEL. ONLY MAG.BUB.#1
E60E B7 AC00     STA A      BCDR      WRITE MSB
E611 39          RTS
*
**  END  **
*
*
E612 WRX4 EQU   *      WRITE REGISTERS VERSION 4
*
* - A ROUTINE TO WRITE BMC REGISTERS IN
*   PREPERATION FOR A READ COMMAND
* - REGISTER SET UP:(SEE BUBBLE SYSTEM DOCUMENTATION)
*   BLR=$1001,2 FSA CH., 1 PAGE/BUBBLE DATA TRANSFER
*   ER =$49, NO PARITY INT., ERROR INTERRUPTS ONLY ON
*               TIMING AND UNCORR. ERR., COMMAND INT.,

```



```

*          BOOTLOOP MASKED,MFBTR HIGH,NO DMA
*  AR=%00000SSS,$SS , SELECT BUBBLE CHIP #1 ONLY,
*          S,S SELECTES DESIRED PAGE
*  RAC=$00 , POINTS TO FIFI ON EXIT
* - INPUTS:- ADDRESS OF PAGE TO BE ACCESSED(TWO BYTES)
*          IS TO BE PUSHED ON STACK BEFORE ENTERY
* - NO OUTPUTS
* - CHANGES A,X,CCR
*
E612 86 0B          LDA A   #$0B
E614 B7 AC01        STA A   BCSR
E617 86 01          LDA A   #$01          WRITE BLR LSB
E619 B7 AC00        STA A   BCSR
E61C 86 10          LDA A   #$10          WRITE BLR MSB
E61E B7 AC00        STA A   BCSR
E621 86 49          LDA A   #$49          WRITE ER
E623 B7 AC00        STA A   BCSR
E626 30            TSX
E627 A6 03          LDA A   3,X          GET ADDRESS LSB
E629 B7 AC00        STA A   BCSR          WRITE ADDRESS LSB
E62C A6 02          LDA A   2,X          GET ADD. MSB
E62E 84 07          AND A   #$07          SEL.ONLY MAG.BUB. #1
E630 B7 AC00        STA A   BCSR          WRITE MSB
E633 39            RTS
*
**  END  **
*
*          TABLES
E721                                ORG      $E721
*****
*
*
E721 7E E30D VCALP  JMP      CALP
*
*
E724 FILTA EQU * FILTER A , LPF, T=.025SEC, FC=3.1 HZ.
E724 062C      FDB      $062C,$0C58,$062C,$40CD,$A683
E726 0C58
E728 062C
E72A 40CD
E72C A683
E72E 0517      FDB      $0517,$0A2E,$0517,$1EFA,$CCAB
E730 0A2E
E732 0517
E734 1EFA
E736 CCAB
*
*
E738 DDATAL EQU *      DISPLAY "DATA LOST"
*
E738 1F          FCB      $1F          "D"
E739 77          FCB      $77          "A"
E73A 45          FCB      $45          "T"

```


E73B	F7	FCB	\$F7	"A."
E73C	38	FCB	\$38	"L"
E73D	6E	FCB	\$6E	"S"
E73E	C5	FCB	\$C5	"T."
E73F	00	FCB	\$00	" "

*

*

E740	D104TB EQU *	DISPLAY "10 4" TABLE
E740	05 FCB	\$05,\$7D,0,\$27,0,0,0,0
E741	7D	
E742	00	
E743	27	
E744	00	
E745	00	
E746	00	
E747	00	

*

*

E748	52	FORTON FCC	' R PROCESS 6=R(*L+1)'
E749	20		
E74A	50		
E74B	52		
E74C	4F		
E74D	43		
E74E	45		
E74F	53		
E750	53		
E751	20		
E752	36		
E753	3D		
E754	52		
E755	28		
E756	2A		
E757	4C		
E758	2B		
E759	31		
E75A	29		
E75B	0D	FCB	\$0D,\$04
E75C	04		
E75D	43	COPYON FCC	' COPY *SOURCE* TO R(*L+1)'
E75E	4F		
E75F	50		
E760	59		
E761	20		
E762	2A		
E763	53		
E764	4F		
E765	55		
E766	52		
E767	43		
E768	45		
E769	2A		
E76A	20		
E76B	54		


```

E76C 4F
E76D 20
E76E 52
E76F 28
E770 2A
E771 4C
E772 2B
E773 31
E774 29
E775 0D      FCB      $0D,$04
E776 04
E777 12      RDYTBL  FCB      %00010010      R
E778 7A      FCB      %01111010      E
E779 77      FCB      %01110111      A
E77A 1F      FCB      %00011111      D
E77B 2F      FCB      %00101111      Y
E77C 00      FCB      0,0,0      3 BLANKS
E77D 00
E77E 00
*
*
E77F FILT2 EQU *  FILTER 2 , LPF, T=.025SEC, FC=.6HZ
E77F 0046      FDB      $0046,$008C,$0046,$7601,$88E6
E781 008C
E783 0046
E785 7601
E787 88E6
E789 0043      FDB      $0043,$0086,$0043,$6A7A,$947A
E78B 0086
E78D 0043
E78F 6A7A
E791 947A
*
*
E793 FILT4 EQU *  FILTER 4 , LPF, T=.025SEC, FC=1.2HZ
E793 011A      FDB      $011A,$0234,$011A,$6A24,$9175
E795 0234
E797 011A
E799 6A24
E79B 9175
E79D 0101      FDB      $0101,$0202,$0101,$558C,$A670
E79F 0202
E7A1 0101
E7A3 558C
E7A5 A670
*
*
E7A7 FILT8 EQU *  FILTER 8 , LPF, T=.025SEC, FC=2.4HZ
E7A7 0418      FDB      $0418,$0830,$0418,$4F78,$A029
E7A9 0830
E7AB 0418
E7AD 4F78
E7AF A029
E7B1 037A      FDB      $037A,$06F4,$037A,$302C,$C1ED

```



```

E7B3 06F4
E7B5 037A
E7B7 302C
E7B9 C1ED
*
E7BB FILTBP EQU *    COEFFECIENTS FOR FILTER BYPASS
E7BB 7FFF      FDB    $7FFF,$0000,$0000,$8000,$0000
E7BD 0000
E7BF 0000
E7C1 8000
E7C3 0000
E7C5 7FFF      FDB    $7FFF,$0000,$0000,$8000,$0000
E7C7 0000
E7C9 0000
E7CB 8000
E7CD 0000
*
*
*   ROM6 JUMP TABLE
*
E7D0                                ORG      $E7D0
*
E7D0 7E E16C      JMP      GTBT
E7D3 7E E5F0      JMP      WRX3
E7D6 7E E5D8      JMP      DSCT
E7D9 7E E3A1      JMP      BDB
E7DC 7E E0A9      JMP      SPTP
E7DF 7E E08D      JMP      CFOF
E7E2 7E E196      JMP      FLAG
E7E5 7E E154      JMP      FDID
E7E8 7E E293      JMP      CAL
E7EB 7E E612      JMP      WRX4
END

```


ROM7

M68SAM is the property of Motorola Spd, Inc.
Copyright 1974 by Motorola Inc.

Motorola M6800 Cross Assembler, Release 1.1

```

NAM      ROM7      BUBBLE MEMORY SYSTEM ROUTINES
*
*
*
*
*      BUBBLE MEMORY CONTROLLER ADDRESSES
*
AC04 BICS EQU $AC04 - BUBBLE INTERRUPT CONTROL/STATUS
AC01 BCSR EQU $AC01 - BUBBLE CONTROL/STATUS REGISTER
AC00 BCDR EQU $AC00 - BUBBLE CONTROLER DATA REGISTER
07FD BMSA EQU $07FD - ADDRESS OF BMS STATUS PAGE
*                      (HIGHEST PAGE, #2045)
*
*
*      BMC  POWER FAIL OUTPUT  ( CB2 ON PIA #4 )
*
A003 LPCRL EQU $A003 - LOW CONTROL REG.      -
*
*
*      VARIABLES USED ONLY BY ROM 7
*
0055                      ORG      $0055
*
0055 0002 BMWA  RMB 2  BUBBLE MEMORY WRITE (PAGE) ADD.
0057 0002 BMRA  RMB 2  BUBBLE MEMORY READ (PAGE) ADD.
0059 0001 BSS   RMB 1  BUBBLE SYSTEM STATUS  $00 EMPTY,
*                      $10 FULL, $01 NOT EMPTY
005A 0001 BMIBS RMB 1  BUBBLE MEMORY INPUT BUFFER STATUS
*                      BIT 7=1 WRITE ACTIVE,BIT 6=1 WRITE
*                      STARTED,BIT 5=1 WRITE REQUESTED
005B 0001 BMOBS RMB 1  BUBBLE MEMORY OUTPUT BUFFER STATUS
*                      AS ABOVE BUT FOR READ,READ ACTIVE AND
*                      READ REQUESTED ARE MUTUALLY EXCLUSIVE
005C 0002 BMSIX RMB 2  STORAGE LOC. FOR X IN ROUTINES
005E 0002 BMSOX RMB 2  BMSI, BMSO
0060 0001 BMIBWR RMB 1  BMIB WRITE REQUEST
0061 0001 BMIBC  RMB 1  INPUT COUNTER
0062 0002 BMIBE  RMB 2  END PTR. TO BMIB
0064 0002 BMIBB  RMB 2  BEGINNING PTR. TO BMIB
0066 0002 BMIBBT RMB 2
0068 0001 BMWC   RMB 1  BMIB WRITE COUNTER
0069 0002 BMOBE  RMB 2  END TO BMOB
006B 0002 BMOBET RMB 2
006D 0002 BMOBB  RMB 2  BEGINNING PTR. TO BMOB
006F 0001 BMOBC  RMB 1  BMOB COUNTER (# BYTES IN BMOB)
0070 0001 BMOBCT RMB 1
0071 0001 BMRC   RMB 1  BUBBLE MEMORY READ COUNTER

```



```

0072 0001 BMSIB   RMB 1   BMS INPUT BYTE
0073 0001 BSIE1   RMB 1   BMS INIT. ERRORS SINCE POWER UP
0074 0001 BCPUE1  RMB 1   BMS PWR.UP ERRORS SINCE PWR. UP

```

```

*
```

```

0800 BMIB      EQU $0800      BUBBLE MEMORY INPUT BUFFER
08C0 BMIBST    EQU BMIB+192    BMIB STOP (192 BYTES)
08C0 BMOB      EQU BMIBST      BUBBLE MEMORY OUTPUT BUFFER
0940 BMOBST    EQU BMOB+128    BMOB STOP (128 BYTES)
0650 BSIL      EQU $0650      BUBBLE SYSTEM INPUT LINE (64 B)
0690 BSOL      EQU $0690      BUBBLE SYSTEM OUTPUT LINE(64 B)

```

```

*
```

```

*
```

```

*
```

BMS ERROR COUNTERS

```

*
```

```

0940                      ORG      $0940

```

```

*
```

```

0940 0001 FNEW      RMB 1      FIFO NOT EMPTY ON WRITE
0941 0001 FNER      RMB 1      FIFO NOT EMPTY ON READ
0942 0001 BSIE      RMB 1      BUBBLE SYSTEM INITIALIZATION
0943 0001 BCPUE     RMB 1      BUBBLE CONTROLLER POWER UP
0944 0001 BCIHE1    RMB 1
0945 0001 BCIHE2    RMB 1      BCIH ERRORS
0946 0001 BCIHE3    RMB 1
0947 0001 BDIHE4    RMB 1      BDIH ERROR
0948 0001 WOFE      RMB 1      WRITE OP. FAIL
0949 0001 WTE       RMB 1      WRITE TIMING
094A 0001 WCE       RMB 1      WRITE CORRECTABLE ERROR
094B 0001 WUCE      RMB 1      WRITE UNCORRECTABLE ERROR
094C 0001 ROFE      RMB 1      READ OP. FAIL
094D 0001 RTE       RMB 1      READ TIMING
094E 0001 RCE       RMB 1      READ CORRECTABLE ERROR
094F 0001 RUCE      RMB 1      READ UNCORRECTABLE ERROR

```

```

*
```

```

*
```

TEST ROUTINE COUNTERS

```

*
```

```

0950 0002 WPC      RMB 2      WRITE PAGE COUNTER
0952 0002 WSC      RMB 2      WRITE SEQUENCE COUNTER
0954 0001 WLC      RMB 1      WRITE LINE COUNTER
0955 0001 WBC      RMB 1      WRITE BYTE COUNTER
0956 0002 RPC      RMB 2      READ PAGE COUNTER
0958 0002 RSC      RMB 2      READ SEQUENCE COUNTER
095A 0001 RLC      RMB 1      READ LINE COUNTER
095B 0001 RBC      RMB 1      READ BYTE COUNTER

```

```

*
```

```

*
```

```

*
```

```

*
```

MONITOR GLOBAL VARIABLES AND FLAGS

```

*
```

```

0000      READEN EQU      $0000 - READ ENABLE
0004      ENTRY1 EQU      $0004 - ENTRIES VIA ENTR FUNCTION
0005      ENTRY2 EQU      $0005
0006      ENTRY3 EQU      $0006
0007      ENTRY4 EQU      $0007
0008      ENTRY5 EQU      $0008

```



```

0009      ENTRY6 EQU      $0009
000A      ENTRY7 EQU      $000A
000B      ENTRY8 EQU      $000B
000D      IXRV   EQU      $000D,E   X REG. STORAGE,MAIN LINES
0012      LEDNO  EQU      $0012 THE # OF THE LED TO BE
*          WRITTEN INTO NEXT

```

```

*
* LINKAGE TABLE ENTRIES FOR SWI  ROUTINES
*

```

```

F40C      IAITX  EQU      $F40C
F40E      IRX2   EQU      $F40E
F410      IPSHX  EQU      $F410

```

```

*
* SUBROUTINE LABLES
*

```

```

FFCA      DSPSN  EQU      $FFCA
FFCD      DSPDP  EQU      $FFCD
FFD0      CLDISP EQU      $FFD0
FFD6      DTSR2  EQU      $FFD6
FFDF      DSPXAB EQU      $FFDF
FFC1      READ2  EQU      $FFC1
FFBB      WRITE0 EQU      $FFBB
FFC4      WRITE2 EQU      $FFC4
FFE8      READ4  EQU      $FFE8
FFEB      WRITE4 EQU      $FFEB
FFF1      DELAY2 EQU      $FFF1
FBF6      SCNDSR EQU      $FBF6
FFC7      RDWL   EQU      $FFC7
FFBE      FLSH   EQU      $FFBE
F3F1      TMSR   EQU      $F3F1
E7D3      WRX3   EQU      $E7D3
E7EB      WRX4   EQU      $E7EB
E7D6      DSCT   EQU      $E7D6
BFE5      SYIDWL EQU      $BFE5

```

```

*
* SOME TABLES USED BY ROM 7
*

```

```

E777      RDYTBL EQU      $E777
E740      D104TB EQU      $E740

```

```

*
* CONSTANTS USED BY ROM7

```

```

0080      FRFR   EQU      $80
0081      FREN   EQU      $81

```

```

*
*
* B000          ORG      $B000

```

```

*****

```

```

*
* B000 RM7RST EQU * RESET FOR ROM 7 ( THE BMS )

```



```

*
* - TO BE USED AFTER POWERUP
* - IF POWER IS ALREADY UP AND RESET IS EXECUTED,
*   ALL ENTERED INTO BMS IS EFFECTIVLY LOST SINCE THE
*   LAST TIME BMSD WAS RUN AS THE BMS STATUS IS RESET
* - BMC IS GIVEN A POWER FAIL PULSE UNTIL OP.COMPLETE
*   STATUS IS RETURNED
*   ( USES CB2 ON PIA #4 (LPCRL) TO POWERFAIL BMC )
* - INITIALIZES BMS STATUS BY RESTORING BSS,BMWA,BMRA
*   TO THEIR VALUES BEFORE BMSD WAS RUN
*
B000 86 FF          LDA A  #$FF
B002 97 73          STA A  BSIE1  INIT.BMS INIT.ERR.CTR
B004 97 74          STA A  BCPUE1 INIT.BMS PWR.UP ERR.CTR
B006 0E             CLI      ENABLE INTERRUPTS FOR MONITOR
*
B007 C6 34          RM7RL1 LDA B  #$34
B009 F7 A003        STA B  LPCRL   SEND PWR FAIL TO BMC
B00C BD FFF1        JSR      DELAY2
B00F 0200           FDB      $0200  DELAY 1 SEC.
B011 C6 04          LDA B  #$04
B013 F7 A003        STA B  LPCRL   SEND POWER OK TO BMC
B016 BD FFF1        JSR      DELAY2
B019 0400           FDB      $0400  DELAY 2 SEC.
B01B 7F AC01        CLR      BCSR   WRITE TO A BMC REG.
*                      REQ. FOR STATUS READ AFTER POWER UP
B01E B6 AC01        LDA A  BCSR
B021 84 FD          AND A  #$FD   IGNORE PARITY ERRORS
B023 81 40          CMP A  #$40   DO UNTIL OP. COMPLETE
B025 27 05          BEQ      RM7RS1 SKIP IF COMPLETE
B027 7A 0074        DEC      BCPUE1 LOG POWER UP ERROR
B02A 20 DB          BRA      RM7RL1
*
B02C BD B4F6 RM7RS1 JSR      BMSU   START UP BMS
*
B02F CE E777        LDX      #RDYTBL
B032 BD FFD6        JSR      DTSR2  DISPLAY "READY"
*
B035 7E BFE5        JMP      SYIDWL  SYSTEM IDLE LOOP
*
**  END  **
*
B038 000C           RMB      12
*
B044 BCIH EQU *    BUBBLE COMMAND INTERRUPT HANDLER
*
* - SERVICES INTERRUPTS FROM THE BMC REQUESTING THE
*   NEXT COMMAND, (BMC NOT BUSY)
* - SERVICES AND LOGS COMMAND EXECUTION ERRORS
* - HAS LOWER PRIORITY THAN BDIH
* - ASSUMES THE BMC IS NOT BUSY
*
B044 B6 AC01        LDA A  BCSR   IS STATUS OF BMC
B047 88 40          EOR A  #$40   OP. COMPLETE,NO FAIL,

```



```

B049 85 7C          BIT A  #$7C    NO T.ERROR,NO READ E.
B04B 27 03          BEQ      BCIHS0  SKIP IF TRUE
B04D 7E B0B2        JMP      BCIHSE  JUMP TO ERROR SERVICE
*
B050 96 5A          BCIHS0 LDA A   BMIBS  CHECK FOR WRITE ACTIVE
B052 2A 2E          BPL      BCIHSR  SKIP IF NOT ACTIVE
*
B054 85 40          BIT A   #$40    WRITE STARTED?
B056 26 04          BNE      BCIHC2  GO TO CASE 2 IF SO
*
B058 BCIHC1 EQU *   CASE 1, WRITE ACTIVE BUT NOT STARTED
*
B058 BD B227        JSR      BMWS    START WRITE PROCESS
B05B 3B             RTI          EXIT
*
B05C BCIHC2 EQU *   CASE 2 , WRITE ACITIVE AND STARTED
*                   ( WRITE SHOULD BE COMPLETE )
B05C BD B29F        JSR      BMWE    END WRITE PROCESS
B05F 96 5B          LDA A   BMOBS   CHECK FOR READ REQ.
B061 27 0B          BEQ      BCIHS4  SKIP IF NONE
B063 86 80          LDA A   #$80    IND. READ ACTIVE,NOT
B065 97 5B          STA A   BMOBS   STARTED OR REQUESTED
B067 96 5A          LDA A   BMIBS   WRITE NOT ACT.NOT ST.
B069 84 3F          AND A   #$3F    DO NOT ALTER REQUEST
B06B 97 5A          STA A   BMIBS   BIT
B06D 3B             RTI          EXIT
*
B06E 96 5A          BCIHS4 LDA A   BMIBS   WRITE REQUEST?
B070 85 20          BIT A   #$20
B072 27 05          BEQ      BCIHS5  SKIP IF NONE
B074 86 80          LDA A   #$80    INDICATE WRITE ACT.
B076 97 5A          STA A   BMIBS   NOT STARTED, NO REQ.
B078 3B             RTI          EXIT
*
B079 7F 005A        BCIHS5 CLR      BMIBS  WRITE NOT ACT.NOT ST.
B07C 86 00          LDA A   #$00    NO REQ;DISABLE BMC INT.
B07E B7 AC04        STA A   BICS
B081 3B             RTI          EXIT
*
B082 96 5B          BCIHSR LDA A   BMOBS   CHECK FOR READ ACT.
B084 2A 23          BPL      BCIHC5  DO CASE 5 IF NOT
B086 85 40          BIT A   #$40    READ STARTED?
B088 26 04          BNE      BCIHC4  DO CASE 4 IF STARTED
*
B08A BCIHC3 EQU *   CASE 3 , READ ACTIVE BUT NOT STARTED
B08A BD B31E        JSR      BMRS    START READ PROCESS
B08D 3B             RTI          EXIT
*
B08E BCIHC4 EQU *   CASE 4 , READ ACTIVE AND STARTED
*                   (READ SHOULD BE COMPLETE)
B08E BD B386        JSR      BMRE    END READ PROCESS
B091 7F 005B        CLR      BMOBS   IND. READ INACT,UNREQ.
B094 96 5A          LDA A   BMIBS   WRITE REQUESTED?
B096 85 C0          BIT A   #$C0    IF WRITE ACT.OR ST.

```



```

B098 26 0F          BNE      BCIHC5    DO CASE 5 (LOGIC E)
B09A 85 20          BIT  A    #$20     CHECK FOR WRITE REQ.
B09C 27 05          BEQ      BCIHS9    SKIP IF NO REQUEST
*
B09E 86 80          LDA  A    #$80     IND.WRITE ACT,NOT ST.
B0A0 97 5A          STA  A    BMIBS    NO REQUEST
B0A2 3B             RTI              EXIT
*
B0A3 86 00          BCIHS9 LDA  A    #$00     DISABLE BMC INT.
B0A5 B7 AC04        STA  A    BICS
B0A8 3B             RTI              EXIT
*
B0A9              BCIHC5 EQU *      CASE 5 , BCIH LOGIC ERROR
B0A9 7A 0944        DEC      BCIHE1    LOG ERROR
B0AC 86 00          LDA  A    #$00
B0AE B7 AC04        STA  A    BICS      DISABLE BMC INT.
B0B1 3B             RTI              EXIT
*
B0B2              BCIHSE EQU *      SOME BMC COMMAND EXECUTION ERROR
*                      (A COMMAND JUST COMPLETED OR A NEW
*                      COMMAND TO BE SENT)
B0B2 88 08          EOR  A    #$08     CHECK FOR OP COMP.
B0B4 85 7C          BIT  A    #$7C     CORR.ERR., NO OTHERS
B0B6 27 0B          BEQ      BCIHSN    SKIP IF TRUE
B0B8 88 0C          EOR  A    #$0C     OP. COMP.,UNCORR.
B0BA 85 7C          BIT  A    #$7C     AND NO OTHER ERRORS?
B0BC 26 21          BNE      BCIHSA    SKIP IF NOT TRUE
B0BE 7A 094F        DEC      RUCE      LOG UCORR. ERROR
B0C1 20 10          BRA      BCIHSC
*
B0C3 D6 5B          BCIHSN LDA  B    BMOBS    CHECK IF READ ACTIVE
B0C5 2A 05          BPL      BCIHSB    SKIP IF NOT
B0C7 7A 094E        DEC      RCE      LOG A READ CORR.ER.
B0CA 20 07          BRA      BCIHSC
B0CC D6 5A          BCIHSB LDA  B    BMIBS    CHECK IF WRITE ACT.
B0CE 2A 06          BPL      BCIHLE    SKIP IF NOT
B0D0 7A 094A        DEC      WCE      LOG WRITE CORR.ERR.
B0D3 7E B050        BCIHSC JMP      BCIHS0    LOOK FOR NEXT COMMAND
*
B0D6 7A 0945        BCIHLE DEC      BCIHE2
B0D9 86 00          LDA  A    #$00
B0DB B7 AC04        STA  A    BICS      DISABLE BMC INT.
B0DE 3B             RTI              EXIT
*
B0DF BCIHSA EQU *      OP. FAIL OR OP. TERMINATED
B0DF B6 AC01        LDA  A    BCSR      GET BMC STATUS
B0E2 D6 5A          LDA  B    BMIBS    CHECK IF WRITE ACT.
B0E4 2A 27          BPL      BCIHSD    SKIP IF NOT
B0E6 85 20          BIT  A    #$20     CHECK FOR OP. FAIL
B0E8 27 05          BEQ      BCIHSI
B0EA 7A 0948        DEC      WOFE
B0ED 8D 50          BSR      BCIHDF    DIS. FAIL
B0EF 85 10          BCIHSI BIT  A    #$10    CHECK FOR TIMING ERR.
B0F1 27 05          BEQ      BCIHSF

```



```

B0F3 7A 0949          DEC      WTE
B0F6 8D 47           BSR      BCIHDF      DIS. FAIL
B0F8 85 08      BCIHSF BIT  A  #$08      CHECK FOR CORR. ERR.
B0FA 27 05           BEQ      BCIHSG
B0FC 7A 094A          DEC      WCE
B0FF 8D 3E           BSR      BCIHDF      DIS. FAIL
B101 85 04      BCIHSG BIT  A  #$04      CHECK FOR UNCORR.ERR.
B103 27 05           BEQ      BCIHSH
B105 7A 094B          DEC      WUCE
B108 8D 35           BSR      BCIHDF      DIS. FAIL
B10A 7E B058 BCIHSH JMP      BCIHC1      RESTART WRITE PROCESS
*
B10D D6 5B      BCIHSD LDA  B  BMOBS      CHECK IF READ ACTIVE
B10F 2A 25           BPL      BCIHX1      IF NOT EXIT
B111 85 20           BIT  A  #$20      CHECK FOR OP. FAIL
B113 27 05           BEQ      BCIHSJ
B115 7A 094C          DEC      ROFE
B118 8D 25           BSR      BCIHDF      DIS. FAIL
B11A 85 10      BCIHSJ BIT  A  #$10      CHECK FOR TIMING ERR.
B11C 27 05           BEQ      BCIHSK
B11E 7A 094D          DEC      RTE
B121 8D 1C           BSR      BCIHDF      DIS. FAIL
B123 85 08      BCIHSK BIT  A  #$08      CHECK FOR CORR.ERR.
B125 27 03           BEQ      BCIHSL
B127 7A 094E          DEC      RCE
B12A 85 04      BCIHSL BIT  A  #$04      CHECK FOR UNCORR.ERR.
B12C 27 05           BEQ      BCIHSM
B12E 7A 094F          DEC      RUCE
B131 8D 0C           BSR      BCIHDF      DIS. FAIL
B133 7E B08A BCIHSM JMP      BCIHC3      RESTART READ PROCESS
*
B136 7A 0946 BCIHX1 DEC      BCIHE3      LOG LOGIC ERROR #3
B139 86 00          LDA  A  #$00      NEITHER RD.NOR WRT.ACT.
B13B B7 AC04          STA  A  BICS      DISABLE BMC INT.
B13E 3B             RTI
*
      B13F      BCIHDF EQU      *      BMS FAILURE
B13F CE B146          LDX      #BSFTBL  GET DISPLAY PATTERN
B142 BD FFD6          JSR      DTSR2    DISPLAY "B.S. FAIL"
B145 39             RTS
*
**      END      **
*
B146 BSFTBL EQU *      BMS FAILURE MESSAGE TABLE
*
B146 3E             FCB      $3E      "B."
B147 EE             FCB      $EE      "S."
B148 00             FCB      $00      " "
B149 72             FCB      $72      "F"
B14A 77             FCB      $77      "A"
B14B 05             FCB      $05      "I"
B14C 38             FCB      $38      "L"
B14D 00             FCB      $00      " "
**      END      **

```



```

*
*
B14E BDIH EQU * BUBBLE DATA INTERRUPT HANDLER
*
* - SERVICES DATA REQUEST INTERRUPTS FORM THE BMC
* - WHEN ENABLED HAS HIGHER PRIORITY THAN BCIH
*
B14E D6 5A          LDA B   BMIBS      CHECK IF WRITE
B150 C4 C0          AND B   #$C0      ACTIVE AND STARTED
B152 C8 C0          EOR B   #$C0
B154 26 04          BNE          BDIHS1  SKIP IF NOT
B156 BD B286        JSR          BMW      WRITE PART OF A PAGE
B159 3B             RTI                EXIT
*
B15A D6 5B          BDIHS1 LDA B   BMOBS  CHECK IF READ ACTIVE
B15C C4 C0          AND B   #$C0      AND STARTED
B15E C8 C0          EOR B   #$C0
B160 26 04          BNE          BDIHLE  SKIP IF NOT,(LOGIC E)
B162 BD B354        JSR          BMR      READ PART OF A PAGE
B165 3B             RTI                EXIT
*
B166 7A 0947 BDIHLE DEC          BDIHE4  LOG ERROR
B169 86 01          LDA A   #$01
B16B B7 AC04        STA A   BICS        DISABLE BMC DATA INT.
B16E 3B             RTI                EXIT
*
**  END  **
*
B16F 0006          RMB      6
*
B175 BMSI EQU * BUBBLE MEMORY SYSTEM INPUT
*
* - THE ROUTINE USED TO INPUT DATA INTO THE BMS
*   WHERE THE BMS IS USED AS A LARGE FIFO
* - TAKES 64 OR FEWER DATA BYTES FROM AN ARBITRARY
*   LOCATION AND PLACES THEM IN BMIB (BUBBLE
*   MEMORY INPUT BUFFER)
* - ACTIVATES THE WRITE PROCESS OR SETS A WRITE
*   REQUEST WHEN ANOTHER COMPLETE PAGE HAS ALREADY
*   BEEN ENTERED INTO BMIB
* - IF THE WRITE PROCESS IS ALREADY ACIVATED AS
*   WELL AS REQUESTED,(BMIB CONTAINS AT MOST
*   2 PAGES),NO DATA WILL BE TRANSFERED TO BMIB
* - ONCE THE WRITE PROCESS IS ACTIVE, DATA IS
*   WRITTEN INTO THE BMS A PAGE AT A TIME,(64 BYTES)
* - NOT A REENTRANT ROUTINE BUT ALLOWS INTERRUPTS
*   FOR THE MOST PART,( RX,TX,BMR,BMW )
* - INPUTS : B = # OF DATA BYTES TO BE INPUT TO BMS
*            X = LOCATION OF FIRST DATA BYTE
*            BSS,BMIBS,BMOBS,BMIBE,BMIBC,BMWA,BMRA
* - OUTPUTS: V BIT IN CCR=SET IF BMS FULL,CLEAR OTHWS.
*            Z BIT IN CCR=SET IF NO DATA TRANSFERED
*            AND CLEAR IF TRANSFERED
*            BMIBS,BMIBE,BMIBC

```


* - CHANGES A,B,X,CCR,BMIBS,BMIBE,BMIBC,BMIB,BMIBWR
 * - MAY BE IMMEDIATELY FOLLOWED BY THE BMS ROUTINE
 * VIA AN INTERRUPT

```

*
B175 DF 5C          STX      BMSIX      SAVE LOC. OF INPUT
B177 96 59          LDA A    BSS        CHECK IF BMS FULL
B179 2B 41          BMI      BMSIXF     EXIT IF FULL
B17B 96 5A          LDA A    BMIBS
B17D 88 20          EOR A    #$20 (IF)WRITE REQ.BUT NOT ACT.
B17F 85 A0          BIT A    #$A0 (THEN)EXIT AS WE CANT HAVE
B181 27 3D          BEQ      BMSIXN     TWO WRITE REQUESTS
*
B183 96 5A          LDA A    BMIBS      BMS BECOMING FULL?:
B185 88 80          EOR A    #$80 (IF)NO WRITE REQ.BUT ACT.
B187 85 A0          BIT A    #$A0
B189 26 11          BNE      BMSIS1
B18B DE 55          LDX      BMWA        (THEN)BMS FULL IF
B18D 08              INX                BMWA + 1 = BMRA
B18E 8C 07FD        CPX      #BMSA
B191 26 03          BNE      BMSISA     INC. WITH WRAP AROUND
B193 CE 0000        LDX      #$0000
B196 9C 57          BMSISA CPX      BMRA
B198 27 22          BEQ      BMSIXF     EXIT IF FULL
B19A 20 27          BRA      BMSIS2
*
B19C 96 5A          BMSIS1 LDA A    BMIBS (IF)WRITE REQ. AND ACT.
B19E 88 A0          EOR A    #$A0
B1A0 85 A0          BIT A    #$A0
B1A2 26 1F          BNE      BMSIS2
B1A4 DE 55          LDX      BMWA        (THEN) BMS FULL IF
B1A6 08              INX                BMWA + 2 = BMRA
B1A7 8C 07FD        CPX      #BMSA     INC. WITH WRAP AROUND
B1AA 26 03          BNE      BMSISB
B1AC CE 0000        LDX      #$0000
B1AF 08              BMSISB INX
B1B0 8C 07FD        CPX      #BMSA     INC. WITH WRAP AROUND
B1B3 26 03          BNE      BMSISC
B1B5 CE 0000        LDX      #$0000
B1B8 9C 57          BMSISC CPX      BMRA
B1BA 26 04          BNE      BMSIXN     SKIP IF NOT FULL
*
B1BC 86 00          BMSIXF LDA A    #$00    SET Z FOR NO TRANSFER
B1BE 0B              SEV                SET V FOR BMS FULL
B1BF 39              RTS                EXIT FOR BMS FULL
*
B1C0 86 00          BMSIXN LDA A    #$00    Z=1 FOR NO TRANS.,V=0
B1C2 39              RTS                FOR BMS NOT FULL,EXIT
*
B1C3 C1 40          BMSIS2 CMP B    #64    CHECK IF B .GT. 64
B1C5 22 F9          BHI      BMSIXN     EXIT IF SO
B1C7 5D              TST B              CHECK IF B = 0
B1C8 27 F6          BEQ      BMSIXN     EXIT IF SO
*
B1CA 7F 0060        CLR      BMIBWR

```



```

B1CD DE 5C      BMSIL1 LDX      BMSIX      DATA TRANSFER LOOP
B1CF A6 00      LDA A      0,X      GET BYTE
B1D1 08      INX
B1D2 DF 5C      STX      BMSIX      SAVE LOC. OF NEXT
B1D4 DE 62      LDX      BMIBE      AT END OF BMIB
B1D6 A7 00      STA A      0,X      STORE BYTE
B1D8 08      INX
B1D9 8C 08C0    CPX      #BMIBST    INC. END LOC. WITH
B1DC 26 03      BNE      BMSISD    WRAP AROUND
B1DE CE 0800    LDX      #BMIB
B1E1 DF 62      BMSISD STX      BMIBE
B1E3 7A 0061    DEC      BMIBC      ANOTHER BYTE ENTERED
B1E6 26 07      BNE      BMSISE
B1E8 7C 0060    INC      BMIBWR    IND. A PAGE ENTRD
B1EB 86 40      LDA A      #64      RESET INPUT
B1ED 97 61      STA A      BMIBC    BUFFER COUNTER
B1EF 5A      BMSISE DEC B
B1F0 26 DB      BNE      BMSIL1    DO UNTIL ALL TRANS.
*
B1F2 96 60      LDA A      BMIBWR    A WRITE REQUEST?
B1F4 27 22      BEQ      BMSIX1    EXIT,(A PAGE NOT ENTRD)
*
B1F6 07      TPA      SAVE CCR
B1F7 36      PSH A      *** DO NOT ALLOW INT. AS
B1F8 0F      SEI      FLAGS MUST BE CURRENT
B1F9 96 5B      LDA A      BMOBS    CHECK IF READ ACTIVE
B1FB 2A 06      BPL      BMSIS3    SKIP IF NOT
B1FD 86 20      LDA A      #$20    IND. WRITE REQ.,
B1FF 97 5A      STA A      BMIBS    NOT ACT.,NOT STARTED
B201 20 13      BRA      BMSIX2    EXIT
*
B203 96 5A      BMSIS3 LDA A      BMIBS    CHECK IF WRITE ACT.
B205 2A 06      BPL      BMSIS4    SKIP IF NOT
B207 8A 20      ORA A      #$20    INDICATE WRITE REQ.
B209 97 5A      STA A      BMIBS
B20B 20 09      BRA      BMSIX2    EXIT
*
B20D 86 80      BMSIS4 LDA A      #$80    IND.WRITE ACTIVE
B20F 97 5A      STA A      BMIBS    NO REQ.,NOT STARTED
B211 86 01      LDA A      #$01
B213 B7 AC04    STA A      BICS      ENABLE BMC INT.ONLY
*
B216 32      BMSIX2 PUL A      *** RESTORE INT. MASK
B217 06      TAP
B218 86 0F      BMSIX1 LDA A      #$0F    SET V=0(BMS NOT FULL)
B21A 39      RTS      Z=0(DATA TRANSFERED)
*
* FOR 22 BYTES INPUT, 1.680 MSEC
** END **
*
B21B 000C      RMB      12
*
B227 BMWS EQU * BUBBLE MEMORY WRITE START
*
```



```

* - PARTLY FILLS FIFO WITH DATA
* - STARTS THE WRITE PROCESS(THE TRANSFER OF
*   A PAGE OF DATA TO THE BMS)
* - ENABLES THE BMC DATA INTERRUPT (PIA #6-B)
* - INITIALIZES BMIBBT,BMWC
* - SETS BMIBS TO INDICATE WRITE STARTED
* - INPUTS : BMWA,BSS,BMIBB,BMIBS
* - OUTPUTS: BMIBBT,BMWC,BMIBS
*
B227 86 01          LDA A  #$01          IND. BMS NOT MT
B229 97 59          STA A  BSS           AS WRITE STARTED
*
B22B DE 64          LDX      BMIBB      INITIALIZED TEMPORARY
B22D DF 66          STX      BMIBBT     BEG.BUF.PTR.
B22F 96 55          LDA A  BMWA
B231 D6 56          LDA B  BMWA+1
B233 37             PSH B              *** PLACE BMWA ON STACK
B234 36             PSH A              **
B235 BD E7D3        JSR      WRX3       WRITE BMC REGISTERS
B238 31             INS                **
B239 31             INS                *** RESTORE STACK
B23A 86 1D          LDA A  #$1D        SEND FIFO RESET
B23C B7 AC01        STA A  BCSR        COMMAND
B23F 86 40          LDA A  #64         INITIALIZE # OF
B241 97 68          STA A  BMWC        BYTES YET IN PAGE
B243 96 5A          LDA A  BMIBS
B245 8A C0          ORA A  #$C0        INDICATE WRITE
B247 97 5A          STA A  BMIBS      STARTED AND ACT.
B249 86 03          LDA A  #$03        ENABLE BMS DATA INT.
B24B B7 AC04        STA A  BICS        ALSO
*
B24E F6 AC01 BMWSL1 LDA B  BCSR        WAIT UNTIL BMC NOT BUSY
B251 2B FB          BMI      BMWSL1
B253 86 13          LDA A  #$13        SEND WRITE COMMAND
B255 B7 AC01        STA A  BCSR
B258 C6 0E          LDA B  #14        TRANSFER 14 BYTES
B25A BD B25E        JSR      TBBF      FROM BMIB TO FIFO
B25D 39             RTS                (FIFO MT IN 1.12 MSEC.)
*
* .741 MSEC.
*
** END **
*
*
B25E TBBF EQU *    TRANSFER BYTES FROM BMIB TO FIFO
*
* - TRANSFERS A VARIABLE NUMBER OF BYTES FROM BMIB
*   TO THE BMC FIFO (UNTIL A PAGE IN TRANSFERED)
* - INPUTS : B = NO. OF BYTES TO BE TRANS.
*           BMIBBT - TEMPORARY POINTER TO BEGINNING
*                   OF DATA IN BMIB
*           BMWC   - # OF BYTES YET IN PAGE
* - OUTPUTS: Z IN CCR - CLR. IF A COMPLETE PAGE TRANS.
*           B,BMIBBT,BMWC - UPDATED

```



```

* - CHANGES A,B,X,CCR
*
B25E 96 68          LDA A   BMWC      PAGE COMPLETE?
B260 2E 03          BGT     TBBFS0    SKIP IF NOT
B262 86 0F          LDA A   #$0F      IND. PAGE COMPLETE
B264 39             RTS              RETURN
*
B265 DE 66          TBBFS0 LDX      BMIBBT 4/4 LOC. OF NEXT BYTE
B267 A6 00          TBBFL1 LDA A   0,X    5 GET BYTE
B269 B7 AC00        STA A   BCDR    5 TRANS. BYTE
B26C 08             INX              4
B26D 8C 08C0        CPX      #BMIBST 3 INC. PTR. WITH
B270 26 03          BNE      TBBFS1 4 WRAP AROUND
B272 CE 0800        LDX      #BMIB    3
B275 7A 0068 TBBFS1 DEC      BMWC    6 A PAGE TRANSFERED?
B278 26 05          BNE      TBBFS2 4 SKIP IF NOT
B27A DF 66          STX      BMIBBT
B27C 86 0F          LDA A   #$0F      Z=0 IF PAGE TRANS.
B27E 39             RTS
B27F 5A             TBBFS2 DEC B      2 DO UNTIL B = 0
B280 26 E5          BNE      TBBFL1 4/40
*
B282 DF 66          STX      BMIBBT 5 LOC. OF NEXT BYTE
B284 4F             CLR A      2 Z=1 IF PG.NOT TRANS.
B285 39             RTS              5/12
*
* FOR 10 BYTES, .416 MSEC.;FOR 14, .576 MSEC
** END **
*
*
B286 BMW EQU *      BUBBLE MEMORY WRITE
*
* - DRIVEN BY BDIH
* - TRANSFERS A NUMBER OF DATA BYTES FROM BMIB TO
* BMC FIFO
* - DISABLES BMC DATA INT.WHEN PAGE TRANSFER COMPLETE
*
B286 C6 0A          LDA B   #10      TRANSFER 10 BYTES
B288 BD B25E        JSR     TBBF      FROM BMIB TO FIFO
B28B 27 05          BEQ     BMWX1    EXIT IF PG.NOT TRANS.
B28D 86 01          LDA A   #$01      DISABLE BMC DATA INT.
B28F B7 AC04        STA A   BICS
*
B292 39             BMWX1 RTS
*
* .438 MSEC.
*
** END **
*
B293 000C           RMB      12
*
B29F BMW EQU *      BUBBLE MEMORY WRITE END
*
* - THIS ROUTINE IS ENTERED TO END THE SUCESSFUL

```


* WRITING OF A PAGE TO THE BMS
 * - THE BMIB BEGINNING POINTER IS UPDATED AND WELL
 * AS THE NEXT PAGE LOCATION TO BE WRITTEN TO
 * - BSS IS ADJUSTED TO IND. BMS NOT EMPTY OR BMS FULL
 *

```
B29F DE 66          LDX      BMIBBT    UPDATE BMIBB
B2A1 DF 64          STX      BMIBB
B2A3 DE 55          LDX      BMWA      INC.NEXT WRITE PAGE #
B2A5 08             INX              WITH WRAP AROUND
B2A6 8C 07FD        CPX      #BMSA
B2A9 26 03          BNE      BMWES1
B2AB CE 0000        LDX      #$0000
B2AE DF 55          BMWES1 STX      BMWA
```

```
*
B2B0 9C 57          CPX      BMRA      CHECK IF BMS FULL
B2B2 26 05          BNE      BMWES2    SKIP IF NOT
B2B4 86 80          LDA A    #$80      INDICATE BMS FULL
B2B6 97 59          STA A    BSS
B2B8 39             RTS              EXIT
```

```
*
B2B9 86 01          BMWES2 LDA A    #$01
B2BB 97 59          STA A    BSS      IND. BMS NOT EMPTY
B2BD 39             RTS              EXIT
```

*
 * 48 USEC.
 *

** END **

B2BE BMSO EQU * BUBBLE MEMORY SYSTEM OUTPUT

*
 * - THIS ROUTINE IS USED TO READ DATA FROM THE BMS
 * WHEN CONFIGURED AS A LARGE FIFO BUFFER
 * - TAKES A SPECIFIED NUMBER OF DATA BYTES FROM
 * BMOB BUBBLE MEMORY OUTPUT BUFFER) AND PLACES IT
 * IN ANY LOCATION
 * - THIS ROUTINE REMOVES DATA FROM BMOB ONLY WHEN
 * READ IS NOT ACTIVE AND NOT REQUESTED SO BMOBC
 * (INDICATING THE NUMBER OF BYTES IN BMIB) IS NOT
 * ERRONEOUSLY UPDATED(SEE BCIH CASE 4)
 * - INPUTS : B = NO. OF BYTES TO BE TRANSFERED
 * X = LOC. WHERE BYTES ARE TO BE PLACED
 * BMOB,BMOBS,BMIBS,BMIBB,BMOBC
 * - OUTPUTS: V(IN CCR)=1 IF BMS MT (NO DATA TRANS.)
 * C(IN CCR)=1 IF READ IS ACT.(NO DATA TRANS.)
 * Z(IN CCR)=0 IF DATA WAS PROPERLY TRANS.
 * BMOBC,BMOBB,BMOBS
 * - NOT REENTRANT BUT ALLOWS RX,TX,BMR,BMW INTERRUPTS
 * FOR THE MOST PART
 *

```
B2BE 5D             TST B          CHECK IF B = 0
B2BF 26 01          BNE          BMSOS1 SKIP IF NOT,
B2C1 39             RTS          Z=1(NO DATA),V=0(BMS NOT MT),
                                C=0(READ NOT ACT.)
```



```

B2C2 C1 40      BMSOS1 CMP B   #64      CHECK IF B .LE. 64
B2C4 23 04      BLS          BMSOS2    SKIP IF SO
B2C6 86 00      LDA A     #0          V=0,C=0,
B2C8 0C          CLC                  Z=1
B2C9 39          RTS
*
B2CA 96 5B      BMSOS2 LDA A     BMOBS    READ REQ.OR ACTIVE?
B2CC 27 04      BEQ          BMSOS0    SKIP IF NOT
B2CE 86 00      LDA A     #$00    V=0,Z=1(NO DATA TRANS.)
B2D0 0D          SEC                  C=1(READ ACT.OR REQ.)
B2D1 39          RTS
*
B2D2 D1 6F      BMSOS0 CMP B     BMOBC    ENOUGH DATA IN BMOB?
B2D4 22 20      BHI          BMSOSD    SKIP IF NOT ENOUGH
*
B2D6 DF 5E      BMSOL1 STX          BMSOX    SAVE STORAGE LOC.
B2D8 DE 6D      LDX          BMOBB    BEGIN LOC. OF BMOB
B2DA A6 00      LDA A     0,X      GET DATA BYTE
B2DC 08          INX
B2DD 8C 0940     CPX          #BMOBST    INC. LOC. TO NEXT
B2E0 26 03      BNE          BMSOS3    WITH WRAP AROUND
B2E2 CE 08C0     LDX          #BMOB
B2E5 DF 6D      BMSOS3 STX          BMOBB
B2E7 DE 5E      LDX          BMSOX    GET STORAGE LOC.
B2E9 A7 00      STA A     0,X      STORE BYTE
B2EB 08          INX
B2EC 7A 006F     DEC          BMOBC    ONE LESS IN BMOB,
B2EF 5A          DEC B              ONE LESS TO TRANSFER
B2F0 26 E4      BNE          BMSOL1    DO UNTIL B = 0
*
B2F2 86 0F          LDA A     #$0F    V=0,Z=0(DATA TRANS.)
B2F4 0C          CLC                  C=0(READ NOT ACTIVE)
B2F5 39          RTS
*
B2F6 BMSOSD EQU *    GET MORE DATA FROM BMS
B2F6 07          TPA          GET CCR
B2F7 36          PSH A          *** SAVE CCR(INT. MASK)
B2F8 0F          SEI          DO NOT ALLOW INT.
*
B2F9 96 5A      LDA A     BMIBS    WRITE ACTIVE?
B2FB 2A 0A      BPL          BMSOS4    SKIP IF NOT
B2FD 86 20      LDA A     #$20    IND. A READ REQUEST
B2FF 97 5B      STA A     BMOBS
B301 32          PUL A          *** RESTORE CCR(INT. MASK)
B302 06          TAP
B303 0D          SEC                  C=1(READ ACT.OR REQ.)
B304 86 00      LDA A     #$00    V=0(BMS NOT MT)
B306 39          RTS                  Z=1(NO DATA TRANS.)
*
B307 96 59      BMSOS4 LDA A     BSS    CKECK BMS STATUS
B309 26 05      BNE          BMSOS6    SKIP IF BMS NOT EMPTY
B30B 32          PUL A          *** RESTORE CCR(INT. MASK)
B30C 06          TAP
B30D 4F          CLR A              C=0(READ NOT ACT.OR REQ.)

```



```

B30E 0B          SEV      Z=1(NO DATA TRANS.),V=1(BMS MT)
B30F 39          RTS
*
B310 86 80      BMSOS6 LDA A  #$80 IND.READ ACT.,NOT STARTED,
B312 97 5B          STA A  BMOBS      NO REQUEST
B314 86 01          LDA A  #$01      ENABLE BMC INTERRUPT
B316 B7 AC04       STA A  BICS
B319 32          PUL A          *** RESTOR CCR,INT. MASK
B31A 06          TAP
B31B 4F          CLR A  V=0(BMS NOT MT),Z=1(NO TRANS.)
B31C 0D          SEC          C=1(READ ACT. OR REQ.)
B31D 39          RTS
*
*   FOR 22 BYTES INPUT, 1.268 MSEC.
*
**   END   **
*
*
B31E BMRS EQU *   BUBBLE MEMORY READ START
*
* - THIS ROUTINE STARTS THE BMS READ PROCESS WHERE
*   A PAGE OF DATA IS REMOVED
* - IT ACTIVATES THE DATA TRANSFER READ ROUTINES BY
*   INDICATING READ HAS STARTED AND ENABLING THE
*   BMC DATA REQUEST INTERRUPT
* - INPUTS :  BSRA,BCSR(BMC STATUS)
* - OUTPUTS:  BMOBS,BMRC
* - DRIVEN BY BCIH
*
B31E DE 69      LDX      BMOBE      INIT.TEMP.BUF.END PTR.
B320 DF 6B      STX      BMOBET
B322 96 6F      LDA A    BMOBC      INIT.TEMP.BUF.COUNTER
B324 97 70      STA A    BMOBCT
B326 96 57      LDA A    BMRA      BMS READ PAGE ADDRESS
B328 D6 58      LDA B    BMRA+1    ( 2 BYTES )
B32A 37          PSH B          *** PLACE ON STACK
B32B 36          PSH A          **
B32C BD E7EB    JSR      WRX4      WRITE REG. FOR READ
B32F 31          INS          ** RESTORE STACK
B330 31          INS          ***
B331 86 1D      LDA A    #$1D      FIFO RESET COMMAND
B333 B7 AC01     STA A    BCSR
B336 86 40      LDA A    #64      INIT. READ COUNTER
B338 97 71      STA A    BMRC
B33A 86 C0      LDA A    #$C0     IND. READ STARTED,ACT.
B33C 97 5B      STA A    BMOBS    BUT NOT REQUESTED
B33E 86 03      LDA A    #$03     ENABLE BMC DATA REQ.
B340 B7 AC04     STA A    BICS     INTERRUPT AS WELL
*
B343 F6 AC01    BMRSL 1 LDA B    BCSR
B346 2B FB      BMI      BMRSL 1  WAIT UNTIL NOT BUSY
B348 86 12      LDA A    #$12     SEND READ COMMAND
B34A B7 AC01     STA A    BCSR
B34D 39          RTS

```



```

*
*      .151 MSEC.
*
**    END    **
*
B34E 0006          RMB      6
*
B354 BMR EQU *    BUBBLE MEMORY READ
*
* - THIS ROUTINE IS DRIVEN BY A REQUEST FOR DATA
*   INTERRUPT FROM THE BMC
* - TRANSFERS 10 BYTES FROM THE BMC FIFO TO THE BMOB
*
B354 C6 0A          LDA B   #10
B356 BD B35A        JSR     TBFB  TRANS.BYTES,FIFO TO
B359 39             RTS      BUFFER
*
*      .453 MSEC.
*
**    END    **
*
*
B35A TBFB EQU *    TRANSFER BYTES FROM FIFO TO BMOB
*
* - THIS ROUTINE TRANSFERS A VARIABLE NUMBER OF BYTES
*   FROM THE BMC FIFO TO BMOB WHILE MAINTAINING THE
*   APPROPRIATE FLAGS AND COUNTERS
* - INPUTS : B = NO OF BYTES TO BE TRANSFERED
*             BMOBET,BMRC,BMOBCT
* - OUTPUTS: Z IN CCR 0 IF A COMPLETE PAGE JUST TRANS.
*             BMOBET,BMRC,BMOBCT
* - CHANGES A,B,X,CCR
*
B35A 96 71          LDA A   BMRC  EXIT,PAGE ALREADY TRANS.
B35C 2E 03          BGT     TBFB$0  SKIP IF NOT
B35E 86 0F          LDA A   #$0F    IND. PAGE TRANS.
B360 39             RTS
*
B361 DE 6B          TBFB$0 LDX     BMOBET 4/4 GET TEMP END PTR.
*
B363 B6 AC00 TBFB$1 LDA A   BCDR    4 GET BYTE FORM FIFO
B366 A7 00          STA A   0,X     6 STORE IN BUFFER
B368 08             INX          4 INC. PTR. WITH
B369 8C 0940        CPX     #BMOBST 3 WRAP AROUND
B36C 26 03          BNE     TBFB$1  4
B36E CE 08C0        LDX     #BMOB   3
B371 7C 0070 TBFB$1 INC      BMOBCT 6 IND. ONE MORE BYTE
B374 7A 0071        DEC     BMRC    6 ONE LESS IN TRANS.
B377 26 05          BNE     TBFB$2  4 SKIP,TRANS.NOT COMP.
B379 DF 6B          STX     BMOBET (5)
B37B 86 0F          LDA A   #$0F    (2) Z=0(PG TRANS.COMP.)
B37D 39             RTS      (5)
*
B37E 5A            TBFB$2 DEC B      2

```



```

B37F 26 E2          BNE      TBFBL1    4/43 DO UNTIL B=0
*
B381 DF 6B          STX      BMOBET 5  UPDATE END PTR.
B383 86 00          LDA A   #$00    2  Z=1(PG.TRANS.NOT
B385 39             RTS              5/12  COMPLETE)
*
*   FOR 10 BYTES, .446 MSEC.; FOR 14, .618 MSEC.
**   END   **
*
*
B386 BMRE EQU *     BUBBLE MEMORY READ END
*
* - ROUTINE TO END THE READ PROCESS
* - TRANSFERS THE LAST 14 BYTES FROM FIFO TO BMOB
*   TO COMPLETE THE READING OF A PAGE
* - CALLED BY THE BMS INT. HANDLER ROUTINE (BCIH)
*   ON THE SUCCESSFUL COMPLETION OF A READ COMMAND
* - UPDATES BMOBET AND THE ADDRESS OF THE NEXT
*   PAGE TO BE READ FROM THE BMS
* - SETS BSS TO INDICATE BMS NOT EMPTY OR BMS EMPTY
*
B386 C6 0E          LDA B   #14      TRANS. REMAINING BYTES
B388 BD B35A        JSR      TBFBL1
B38B 86 01          LDA A   #$01      DISABLE BMC DATA INT.
B38D B7 AC04        STA A   BICS
*
B390 DE 6B          LDX      BMOBET  UPDATE BMOBE
B392 DF 69          STX      BMOBE
B394 96 70          LDA A   BMOBCT  UPDATE # OF BYTES
B396 97 6F          STA A   BMOBC
*
B398 DE 57          LDX      BMRA      INC. BMS READ PAGE
B39A 08             INX              ADDRESS WITH WRAP
B39B 8C 07FD        CPX      #BMSA    AROUND
B39E 26 03          BNE      BMRES1
B3A0 CE 0000        LDX      #$0000
B3A3 DF 57          BMRES1 STX      BMRA
B3A5 9C 55          CPX      BMWA      CHECK IF BMS EMPTY
B3A7 26 04          BNE      BMRES2    SKIP IF NOT
B3A9 7F 0059        CLR      BSS      INDICATE BMS EMPTY
B3AC 39             RTS
*
B3AD 86 01          BMRES2 LDA A   #$01  IND. BMS NOT EMPTY
B3AF 97 59          STA A   BSS
B3B1 39             RTS
*
*   .694 MSEC
*
**   END   **
*
B3B2 0006          RMB      6
*
B3B8 BMIBI EQU *    BUBBLE MEMORY INPUT BUFFER
*                   INITIALIZATION

```



```

*
* - SETS ALL COUNTERS AND POINTERS TO INDICATE
*   BMIB IS EMPTY
*
B3B8 7F 005A      CLR      BMIBS      WRITE NOT ACT.OR REQ.
B3BB CE 0800      LDX      #BMIB
B3BE DF 64        STX      BMIBB      SET PTRS. TO BEGINING
B3C0 DF 66        STX      BMIBBT     OF BUFFER
B3C2 DF 62        STX      BMIBE
B3C4 86 40        LDA A    #64        SET CTR TO # BYTES
B3C6 97 61        STA A    BMIBC      ENTERED
B3C8 39           RTS
*
**   END   **
*
*
B3C9 BMOBI EQU *   BUBBLE MEMORY OUTPUT BUFFER
*                   INITIALIZATION
*
*
* - SETS ALL POINTERS AND COUNTERS TO INDICATE BMOB
*   IS EMPTY
*
B3C9 7F 005B      CLR      BMOBS     READ NOT ACT.,NOT REQ.
B3CC CE 08C0      LDX      #BMOB     INIT. POINTERS
B3CF DF 6D        STX      BMOBB
B3D1 DF 69        STX      BMOBE
B3D3 DF 6B        STX      BMOBET
B3D5 7F 006F      CLR      BMOBC      NO DATA IN BUFFER
B3D8 7F 0070      CLR      BMOBCT
B3DB 39           RTS
*
**   END   **
B3DC 0006         RMB      6
*
*
B3E2 BMWF EQU *   BUBBLE MEMORY WRITE FINISH
*
* - WRITES THE REMAINING BYTES OF BMIB INTO THE
*   BMS BY COMPLETING THE PAGE IN BMIB WITH FRAME
*   FRONT AND FRAME END BYTES
* - THIS ROUTINE DOES NOT EXIT UNTIL ALL PAGES IN BMIB
*   ARE TRANSFERED,(BMIB EMPTY), OR UNTIL BMS IS FULL
* - THIS ROUTINE REQUIRES THAT INTERRUPTS ARE ENABLED
*
B3E2 96 61        LDA A    BMIBC      PAGE PART IN BMIB
B3E4 81 40        CMP A    #64
B3E6 27 2A        BEQ      BMWFXL     TO EXIT LOOP IF NOT
*
B3E8 96 5A        BMWFL1 LDA A    BMIBS      WAIT UNTIL NO WRITE
B3EA 88 A0        EOR A    #$A0      AND WRITE REQUESTED
B3EC 85 A0        BIT A    #$A0
B3EE 27 F8        BEQ      BMWFL1
*

```



```

B3F0 CE 0650          LDX      #BSIL CLEAR TWO BYTES IN LINE
B3F3 86 00          LDA A    #0
B3F5 C6 00          LDA B    #0
B3F7 A7 00          STA A    0,X
B3F9 08             INX
B3FA E7 00          STA B    0,X
B3FC 08             INX
B3FD 8C 0652        CPX      #BSIL+2    2 BYTES IN LINE
B400 26 F5          BNE      BMWFL3
*
B402 CE 0650 BMWFL2 LDX      #BSIL      LOCATION OF INPUT
B405 C6 02          LDA B    #2          STORE 2 BYTES
B407 BD B175        JSR      BMSI      IN BMS
B40A 29 06          BVS      BMWFXL    SKIP IF BMS FULL
B40C 96 61          LDA A    BMIBC
B40E 81 3F          CMP A    #63      DO UNTIL A COMPLETE
B410 2D F0          BLT      BMWFL2    PAGE ENTERED
*
B412 96 5A BMWFXL LDA A    BMIBS      WAIT UNTIL WRITE NOT
B414 26 FC          BNE      BMWFXL    ACT.,NOT REQ.
B416 BD B3B8        JSR      BMIBI    CLEAR BMIB
B419 39             RTS              EXIT
*
** END **
*
*
B41A BMSE EQU *    BUBBLE MEMORY SYSTEM EMPTY
*
* - A ROUTINE TO SET ALL POINTERS, COUNTERS, AND
*   STATUS REGISTERS TO INDICATE THE BMS IS EMPTY
*
B41A BD B3B8        JSR      BMIBI    INIT. (CLEAR) BMIB
B41D BD B3C9        JSR      BMOBI    INIT. (CLEAR) BMOB
B420 7F 0059        CLR      BSS      IND. BMS EMPTY
B423 CE 0000        LDX      #$0000
B426 DF 55          STX      BMWA     ZERO BMS WRITE PAGE ADD.
B428 DF 57          STX      BMRA     ZERO BMS READ PAGE ADD.
B42A 39             RTS
*
** END **
*
*
B42B BMSEE EQU *    BUBBLE MEMORY SYSTEM ERROR ERASE
*
* - SETS ALL BMS ERROR COUNTERS TO INDICATE
*   NO ERRORS HAVE OCCURED
* - ASSUMES THE LIST OF ERROR COUNTERS STARTS WITH
*   FNEW AND ENDS WITH RUCE
* - ASSUMES COUNTERS ARE SET TO $FF FOR NO ERRORS
*
B42B 86 FF          LDA A    #$FF
B42D CE 0940        LDX      #FNEW
*
B430 A7 00 BMWSEEL STA A    0,X      CLEAR ERROR COUNTER

```



```

B432 08          INX
B433 8C 0950      CPX      #RUCE+1 DO UNTIL ALL CLEARED
B436 26 F8        BNE      BMSEEL
B438 39          RTS
*
**   END   **
B439 0006          RMB      6
*
*
B43F BMSD      EQU *   BUBBLE MEMORY SYSTEM SHUTDOWN
*
* - SAVES THE STATE OF THE BMS
* - WRITES THE FIRST 64 BYTES OF BMIB INTO PAGE
*   2045 OF THE BMS (FOR DEBUG)
* - THE FIRST OF THOSE BYTES ARE BMWA,BMRA,BSS
* - THE NEXT BYTES ARE ALL THE ERROR COUNTERS
* - THE REMAINING BYTES CAN BE ANY PATTERN ENTERED
*   VIA THE MONITOR
* - ASSUMES THE ERROR COUNTERS START WITH FNEW AND
*   END WITH RUCE
*
B43F CE 0800      LDX      #BMIB      DEST. OF BMS STATUS
B442 96 55        LDA A    BMWA      MOVE BMWA
B444 A7 00        STA A    0,X
B446 08          INX
B447 96 56        LDA A    BMWA+1
B449 A7 00        STA A    0,X
B44B 08          INX
B44C 96 57        LDA A    BMRA      MOVE BMRA
B44E A7 00        STA A    0,X
B450 08          INX
B451 96 58        LDA A    BMRA+1
B453 A7 00        STA A    0,X
B455 08          INX
B456 96 59        LDA A    BSS      MOVE BSS
B458 A7 00        STA A    0,X
B45A 08          INX
*
B45B 3F          SWI          *6* PLACE DEST. OF
B45C 0000        FDB          0,IPSHX  ERR.BLOCK ON STACK
B45E F410
B460 CE 0940      LDX      #FNEW      PLACE START LOCATION
B463 3F          SWI          *4* ON STACK
B464 0000        FDB          0,IPSHX
B466 F410
B468 CE 0950      LDX      #RUCE+1    PLACE STOP LOC.
B46B 3F          SWI          *2*
B46C 0000        FDB          0,IPSHX
B46E F410
B470 BD B4CA      JSR      TNDB      TRANSFER ERROR BLOCK
B473 31          INS
B474 31          INS          *2*
B475 31          INS          REMOVE JUNK FROM STACK
B476 31          INS          *4*

```



```

B477 31      INS
B478 31      INS          *6*
*
B479 CE 07FD  LDX      #BMSA  ADD. OF BMS STATUS PAGE
B47C 3F      SWI      *2* IS TO PLACED ON STACK
B47D 0000    FDB      0,IPSHX
B47F F410
B481 BD E7D3  JSR      WRX3      WRITE BMC REGISTERS
B484 31      INS
B485 31      INS      *2* RESTORE STACK
B486 86 1D    LDA A    #$1D    SEND FIFO RESET COMMAND
B488 B7 AC01  STA A    BCSR
B48B B6 AC01  LDA A    BCSR    WAIT UNTIL BMC NOT BUSY
B48E 2B FB    BMI      BMSDL1
B490 BD B3B8  JSR      BMIBI    RESET BMIB POINTERS
B493 86 40    LDA A    #64    TRANSFER PAGE(64 BYTES)
B495 97 68    STA A    BMWC
B497 86 13    LDA A    #$13    SEND WRITE COMMAND
B499 B7 AC01  STA A    BCSR
*
B49C F6 AC01  LDA B    BCSR
B49F 2A FB    BPL      BMSDL4    WAIT UNTIL BUSY

B4A1 DE 66    LDX      BMIBBT    REMAINDER OF PAGE
B4A3 F6 AC01  LDA B    BCSR
B4A6 C5 01    BIT B    #$01    LOOP UNTIL DATA
B4A8 27 F9    BEQ      BMSDL2    CAN BE ENTERED
B4AA A6 00    LDA A    0,X    GET DATA
B4AC B7 AC00  STA A    BCDR    PLACE DATA IN FIFO
B4AF 08      INX
B4B0 7A 0068  DEC      BMWC    DO UNTIL PAGE
B4B3 26 EE    BNE      BMSDL2    TRANSFER COMPLETE
B4B5 39      RTS      EXIT
*
**      END      **
*
*      AN ALTERNATIVE FOR THE CODE FROM
*      BMSDL4 ONWARDS
*
B4B6 C6 18    LDA B    #24    TRANSFER 24 BYTES TO FIFO
B4B8 BD B25E  JSR      TBBF
*
B4BB B6 AC04  LDA A    BICS
B4BE 85 20    BIT A    #$20
B4C0 27 F9    BEQ      BMSDL3    WAIT FOR DATA REQ.
B4C2 C6 14    LDA B    #20    TRANSFER 20 BYTES
B4C4 BD B25E  JSR      TBBF    FROM BMIB TO FIFO
B4C7 27 F2    BEQ      BMSDL3    DO UNTIL PAGE TRANS.
B4C9 39      BMSDX2  RTS      RETURN
*
**      END      **
*
*
B4CA TNDB    EQU *    TRANSFER DATA BLOCK

```



```

*
* A SUBROUTINE TO MOVE A DATA BLOCK
* THE DESTINATION AND THE LOCATIONS OF THE FIRST BYTE
* AND THE STOP BYTE(LAST BYTE + 1) OF THE DATA TO BE
* MOVED ARE PUSHED ON STACK IN THAT ORDER(2 BYTES EACH)
* A REENTRANT SUBROUTINE AS LONG AS DESTINATIONS DO
* NOT OVERLAP
*
B4CA 30          TNDL1 TSX
B4CB EE 04          LDX      4,X  LOC. OF BYTE TO BE MOVED
B4CD A6 00          LDA A    0,X  GET BYTE
B4CF 30          TSX
B4D0 EE 06          LDX      6,X  GET DESTINATION
B4D2 A7 00          STA A    0,X  MOVE BYTE
*
B4D4 30          TSX          INC.DESTINATION
B4D5 A6 06          LDA A    6,X
B4D7 E6 07          LDA B    7,X
B4D9 CB 01          ADD B    #$01
B4DB 89 00          ADC A    #$00
B4DD A7 06          STA A    6,X
B4DF E7 07          STA B    7,X
*
B4E1 A6 04          LDA A    4,X  INC.LOC.OF BYTE TO BE
B4E3 E6 05          LDA B    5,X  MOVED
B4E5 CB 01          ADD B    #$01
B4E7 89 00          ADC A    #$00
B4E9 A7 04          STA A    4,X
B4EB E7 05          STA B    5,X
*
B4ED A1 02          CMP A    2,X  BLOCK TRANS.COMPLETE?
B4EF 26 D9          BNE      TNDL1
B4F1 E1 03          CMP B    3,X  DO UNTIL
B4F3 26 D5          BNE      TNDL1  STOP BYTE REACHED
*
B4F5 39          RTS          EXIT
*
*      FOR 22 BYTES, 2.359 MSEC.
*
**  END  **
*
*
B4F6 BMSU  EQU * BUBBLE MEMORY SYSTEM START UP
*
* - INITIALIZES THE BMS, AS IF JUST AFTER POWER UP
* - EMPTIES FIFO IN BMC CHIP
* - SENDS BMC INITIALIZE COMMAND
* - READS THE BMS STATUS PAGE AND PLACES IT IN BMIB
* - RESTORES BMWA,BMRA,BSS TO THEIR VALUES BEFORE
*   BMSD (SHUTDOWN) WAS RUN
* - CLEARS (INITIALIZES) BMIB,BMOB
* - CAN BE USED TO TEST THE BMS
*
B4F6 B6 AC01 BMSUL1 LDA A    BCSR      BMC NOT BUSY?

```



```

B4F9 85 80      BIT A    #$80
B4FB 26 F9      BNE      BMSUL1  WAIT UNTIL NOT BUSY
*
B4FD B6 AC01    BMSUL3 LDA A    BCSR      CHECK IF FIFO EMPTY
B500 85 01      BIT A    #$01
B502 27 05      BEQ      BMSUL5  SKIP IF EMPTY
B504 F6 AC00    LDA B    BCDR REMOVE A BYTE FROM FIFO
B507 20 F4      BRA      BMSUL3  DO UNTIL EMPTY
*
B509 4F          BMSUL5 CLR A          SELECT BMS PAGE ZERO
B50A 36          PSH A          ***
B50B 36          PSH A          **
B50C BD E7D3     JSR      WRX3      WRITE BMC REGISTERS
B50F 31          INS          **
B510 31          INS          *** GET JUNK OFF STACK
B511 86 11      LDA A    #$11
B513 B7 AC01    STA A    BCSR SEND BMC INIT. COMMAND
B516 F6 AC01    BMSUL4 LDA B    BCSR
B519 C5 80      BIT B    #$80      WAIT UNTIL
B51B 26 F9      BNE      BMSUL4  BMC NOT BUSY
B51D C8 40      EOR B    #$40      INIT. FAILURE?
B51F C5 7C      BIT B    #$7C
B521 27 0A      BEQ      BMSUS2  SKIP IF NO FAILURE
B523 7A 0073    DEC      BSIE1    LOG INITIALIZATION
B526 26 03      BNE      BMSUS3  ERROR
B528 BD B13F    JSR      BCIHDF    DISPLAY "B.S. FAIL"
B52B 20 DC      BMSUS3 BRA      BMSUL5  DO UNTIL INIT. OK
*
B52D CE 07FD    BMSUS2 LDX      #BMSA  BMS STATUS PAGE ADD.
B530 3F          SWI          *2* ON STACK
B531 0000      FDB          0,IPSHX
B533 F410
B535 BD E7D3     JSR      WRX3      WRITE BMC REGISTERS
B538 31          INS
B539 31          INS          *2* RESTORE STACK
B53A CE 08C0    LDX      #BMOB  DEST.OF PG TO BE READ
B53D 86 40      LDA A    #64      # BYTES IN PAGE
B53F 97 71      STA A    BMRC
B541 86 12      LDA A    #$12      SEND READ COMMAND
B543 B7 AC01    STA A    BCSR
*
B546 F6 AC01    BMSUL6 LDA B    BCSR
B549 C5 01      BIT B    #$01      WAIT UNTIL
B54B 27 F9      BEQ      BMSUL6  DATA AVAILABLE
B54D B6 AC00    LDA A    BCDR      GET DATA BYTE
B550 A7 00      STA A    0,X      STORE DATA
B552 08          INX
B553 7A 0071    DEC      BMRC      DO UNTIL
B556 26 EE      BNE      BMSUL6  A PAGE IS READ
*
B558 FE 08C0    LDX      BMOB      RESTORE BMWA
B55B DF 55      STX      BMWA
B55D FE 08C2    LDX      BMOB+2  RESTORE BMRA
B560 DF 57      STX      BMRA

```



```

B562 B6 08C4      LDA A   BMOB+4
B565 97 59        STA A   BSS          RESTORE BSS
*
B567 CE 0940      LDX      #FNEW      PUSH DEST.OF ERROR
B56A 3F           SWI          *6*      BLOCK
B56B 0000         FDB      0,IPSHX
B56D F410
B56F CE 08C5      LDX      #BMOB+5    PUSH START OF ERROR
B572 3F           SWI          *4*      BLOCK
B573 0000         FDB      0,IPSHX
B575 F410
B577 CE 08D5      LDX      #BMOB+1+5+RUCE-FNEW
B57A 3F           SWI      *2*PUSH END LOC. OF ERROR BLOCK
B57B 0000         FDB      0,IPSHX
B57D F410
B57F BD B4CA      JSR      TNDB      TRANSFER ERROR BLOCK
B582 31           INS
B583 31           INS      *2*
B584 31           INS
B585 31           INS      *4*  RESTORE STACK
B586 31           INS
B587 31           INS      *6*
*
B588 D6 74        LDA B   BCPUE1      UPDATE BCPUE
B58A 5C           INC B
B58B FB 0943      ADD B   BCPUE
B58E F7 0943      STA B   BCPUE
B591 D6 73        LDA B   BSIE1      UPDATE BSIE
B593 5C           INC B
B594 FB 0942      ADD B   BSIE
B597 F7 0942      STA B   BSIE
*
B59A BD B3B8      JSR      BMBIBI INIT. BMS I/O BUFFERS
B59D BD B3C9      JSR      BMOBI
B5A0 39           RTS          EXIT
*
**  END  **
*
*
B5A1 SXVM      EQU * SUBROUTINE EXECUTION VIA MONITOR
*
* - THIS ROUTINE ALLOWS THE MONITOR TO EXECUTE ANY
*   SUBROUTINE (ENDING IN RTS)
* - USE:
*       PRESS "CLR"
*       ENTER SUBROUTINE LOCATION
*       PRESS ". " , "ENTR"
*       ENTER LOCATION OF THIS ROUTINE
*       PRESS "GO TO" , "STRT"
* - AFTER EXECUTION A DECIMAL POINT IS
*   DISPLAYED AND A WAIT LOOP IS ENTERED
*
B5A1 DE 04      LDX      ENTRY1      LOC. OF SUBROUTINE
B5A3 AD 00      JSR      0,X          EXECUTE SUBROUTINE
B5A5 0E         CLI          ALLOW INT. FOR MONITOR

```



```

B5A6 86 01          LDA A   #$01
B5A8 97 00          STA A   READEN ALLOW KEYBOARD ENTRIES
B5AA 5F             CLR B           IN FIRST LED DISPLAY
B5AB BD FFCD        JSR      DSPDP   DECIMAL POINT
B5AE 4F             CLR A           PREPARE TO WRITE
B5AF BD FFBB        JSR      WRITE0  TO FIRST LED
B5B2 3E             SXVML1 WAI
B5B3 20 FD          BRA      SXVML1  WAIT LOOP

```

```

*
**  END  **

```

```

B5B5 BWPT EQU * BUBBLE WRITE PAGE TEST

```

```

*
* - A ROUTINE USED TO TEST WRITING INTO THE BMS
* - WRITES A PAGE CONSISTING OF 4 LINES
*   OF 16 BYTES EACH
* - EACH LINE CONSISTS OF THE PAGE NUMBER (2 BYTES)
*   FOLLOWED BY 7 VALUES OF A 2 BYTE INCREMENTING
*   SEQUENCE
* - INPUTS : WPC,WSC
* - OUTPUTS: V(CCR) SET IF BMS FULL(CLEAR OTHERWISE)
*           WSC
* - CHANGES A,B,X,CCR

```

```

B5B5 86 04          LDA A   #$04
B5B7 B7 0954        STA A   WLC    # LINES TO BE WRITTEN
*
B5BA 86 07          BWPTL3 LDA A   #$07  # SEQUENCE COUNTS/LINE
B5BC B7 0955        STA A   WBC
B5BF FE 0950        LDX      WPC
B5C2 FF 0650        STX      BSIL  START LINE WITH PAGE #
B5C5 CE 0652        LDX      #BSIL+2
B5C8 B6 0952        LDA A   WSC      GET SEQUENCE COUNT
B5CB F6 0953        LDA B   WSC+1
B5CE CB 01          ADD B   #$01     INCREMENT
B5D0 89 00          ADC A   #$00

```

```

*
B5D2 A7 00          BWPTL1 STA A   0,X      ADD COUNT TO LINE
B5D4 08             INX
B5D5 E7 00          STA B   0,X
B5D7 08             INX
B5D8 CB 01          ADD B   #$01     INC. SEQ. COUNT
B5DA 89 00          ADC A   #$00
B5DC 7A 0955        DEC      WBC
B5DF 26 F1          BNE      BWPTL1  DO UNTIL LINE OK

```

```

*
B5E1 B7 0952        STA A   WSC      UPDATE WSC
B5E4 F7 0953        STA B   WSC+1
B5E7 CE 0650        BWPTL2 LDX      #BSIL  LOC. OF INPUT DATA
B5EA C6 10          LDA B   #16      NO. OF BYTES
B5EC BD B175        JSR      BMSI    INPUT DATA TO BMS
B5EF 29 08          BVS      BWPTX1  EXIT IF BMS FULL
B5F1 27 F4          BEQ      BWPTL2  REPETE,LINE NOT INPUT

```



```

*
B5F3 7A 0954      DEC      WLC      DO UNTIL
B5F6 26 C2        BNE      BWPTL3   ALL LINES WRITTEN
*
B5F8 0A          CLV              IND. BMS NOT FULL
B5F9 39          BWPTX1 RTS        EXIT
*
**  END  **
B5FA 0006        RMB      6
*
*
B600 BSWT      EQU *  BUBBLE SYSTEM WRITE TEST
*
* - WRITES THE NUMBER OF PAGES IN ENTRY1 INTO THE BMS
* - EACH PAGE CONSISTS OF 64 BYTES (SEE BWPT)
* - IF THE BMS DOES NOT BECOME FULL, BSTX IS EXECUTED
*
B600 96 59      LDA  A  BSS
B602 2B 2B      BMI      BTXF      EXIT IF BMS FULL
*
B604 CE 0000    LDX      #$0000
B607 FF 0950    STX      WPC      CLEAR WRITE PG COUNT
B60A FF 0952    STX      WSC      CLEAR WRITE SEQ.COUNT
B60D BD B42B    JSR      BMSEE     ERASE BMS ERRORS
*
B610 BD B5B5    JSR      BWPT      WRITE A PAGE TO BMS
B613 29 1A      BVS      BTXF      EXIT IF BMS FULL
*
B615 BD FFD0    JSR      CLDISP
B618 B6 0950    LDA  A  WPC      DISP.WRITE PG COUNT
B61B F6 0951    LDA  B  WPC+1
B61E BD FFEB    JSR      WRITE4
B621 FE 0950    LDX      WPC      INCREMENT
B624 08         INX
B625 FF 0950    STX      WPC
B628 9C 04      CPX      ENTRY1   DO UNTIL
B62A 26 E4      BNE      BSWTL1   PAGE COUNT=END PAGE
*
B62C 7E B77A    JMP      BSTX      EXIT
*
*
*
*
B62F 86 04      BTXF      LDA  A  #4      4 CHARACTERS
B631 C6 04      LDA  B  #4      STARTING IN LED #4
B633 CE B63C    LDX      #DBSFT   ARE TO BE WRITTEN
B636 BD E7D6    JSR      DSCT     WITH " B.S.F. "
B639 7E B7AC    JMP      BSTW     STOP
*
*
B63C      DBSFT  EQU *  DISPLAY BUBBLE SYSTEM FULL TABLE
B63C 00      FCB      0          " "
B63D BE      FCB      $BE       "B."
B63E EE      FCB      $EE       "S."

```



```

B63F F2          FCB      $F2      "F."
*
*
B640 0006        RMB      6
*
B646 BSRT      EQU * BUBBLE SYSTEM READ TEST
*
* - READS THE NUMBER OF PAGES STORED IN ENTRY1
*   FROM THE BMS
* - CHECKS TO SEE IF THE DATA READ IS THE SAME AS
*   THAT WRITTEN BY BSWT,(CALLS BRPT)
* - AFTER EACH SUCCESSFUL PAGE READ THE PAGE
*   NUMBER IS DISPLAYED
* - IF AN ERROR IS DETECTED IN THE DATA READ, "RD. ER."
*   IS DISPLAYED AND THE PROGRAM STOPS
* - IF THERE ARE NO READ ERRORS, CONTROL PASSES TO
*   THE BSTX STUB.
*
B646 96 59      LDA A  BSS
B648 27 2B      BEQ    BTXE      EXIT IF BMS EMPTY
B64A CE 0000    LDX    #$0000
B64D FF 0956    STX    RPC      CLEAR READ PAGE COUNT
B650 FF 0958    STX    RSC      CLEAR READ SEQ. COUNT
B653 BD B42B    JSR    BMSEE    ERASE BMS ERRORS
*
B656 BD B68C BSRTL1 JSR    BRPT  READ A TEST PAGE
B659 29 1A      BVS    BTXE      EXIT IF BMS EMPTY
B65B BD FFD0    JSR    CLDISP
B65E B6 0956    LDA A  RPC      DISPLAY PAGE NUMBER
B661 F6 0957    LDA B  RPC+1
B664 BD FFEB    JSR    WRITE4
B667 FE 0956    LDX    RPC      INC. READ PAGE NUMBER
B66A 08         INX
B66B FF 0956    STX    RPC
B66E 9C 04      CPX    ENTRY1    DO UNTIL
B670 26 E4      BNE    BSRTL1    PAGE COUNT=END PAGE #
*
B672 7E B77A    JMP    BSTX      EXIT
*
*
B675 86 04      BTXE    LDA A  #4      4 CHARACTERS
B677 C6 04      LDA B  #4      STARTING AT LED #4
B679 CE B682    LDX    #DBSEMT    ARE "B.S.E."
B67C BD E7D6    JSR    DSCT
B67F 7E B7AC    JMP    BSTW      STOP
*
**   END   **
*
B682 DBSEMT EQU * DISPLAY BUBBLE SYSTEM EMPTY TABLE
B682 00          FCB      0      " "
B683 BE          FCB      $BE    "B."
B684 EE          FCB      $EE    "S."
B685 FA          FCB      $FA    "E."
*

```



```

B686 0006          RMB      6
*
B68C BRPT      EQU *      BUBBLE READ PAGE TEST
*
* - A SUBROUTINE TO BE USED TO TEST READING
*   THE BMS
* - READS A PAGE OF 4 LINES OF 16 BYTES EACH AND
*   CHECKS TO MAKE SURE EACH LINE IS AS WRITTEN BY BWPT
* - ON THE FIRST OCCURANCE OF AN ERROR WE STOP AND
*   "RD.ER." IS DISPLAYED
*
B68C 86 04          LDA A   #$04  INIT. # OF LINES IN PG
B68E B7 095A        STA A   RLC
*
B691 CE 0690 BRPTL1 LDX     #BSOL  GET A LINE FROM BMS
B694 C6 10          LDA B   #16    16 BYTES LONG
B696 BD B2BE        JSR     BMSO
B699 29 3B          BVS     BRPTX1  RETURN IF BMS EMPTY
B69B 27 F4          BEQ     BRPTL1  DO UNTIL DATA TRANS.
*
B69D C6 07          LDA B   #$07  INIT. # OF SEQ.COUNTS
B69F F7 095B        STA B   RBC    TO BE CHECKED
B6A2 FE 0690        LDX     BSOL   CHECK PAGE COUNT
B6A5 BC 0956        CPX     RPC
B6A8 26 2D          BNE     BRPTXE  EXIT IF NOT SAME
B6AA B6 0958        LDA A   RSC    INC. SEQUENCE COUNT
B6AD F6 0959        LDA B   RSC+1
B6B0 CB 01          ADD B   #$01
B6B2 89 00          ADC A   #$00
B6B4 CE 0692        LDX     #BSOL+2 LOC.OF SEQ. COUNTS
*
B6B7 A1 00 BRPTL2  CMP A   0,X     CHECK SEQ. COUNTS
B6B9 26 1C          BNE     BRPTXE  EXIT IF NOT SAME
B6BB 08             INX
B6BC E1 00          CMP B   0,X
B6BE 26 17          BNE     BRPTXE  EXIT IF NOT SAME
B6C0 08             INX
B6C1 CB 01          ADD B   #$01    INC. SEQ. COUNT
B6C3 89 00          ADC A   #$00
B6C5 7A 095B        DEC     RBC
B6C8 26 ED          BNE     BRPTL2  DO UNTIL LINE CHECKED
*
B6CA B7 0958        STA A   RSC    UPDATE RSC
B6CD F7 0959        STA B   RSC+1
B6D0 7A 095A        DEC     RLC    DO UNTIL
B6D3 26 BC          BNE     BRPTL1  ALL LINES CHECKED
*
B6D5 0A             CLV            IND. BMS NOT EMPTY
B6D6 39 BRPTX1      RTS
*
*
B6D7 31 BRPTXE      INS          BMS EMPTY EXIT, REMOVE
B6D8 31             INS          SUBROUTINE RETURN ON STACK
B6D9 86 04          LDA A   #$04  4 CHARACTERS,

```



```

B6DB D6 12          LDA B  LEDNO    IN NEXT LEDS,
B6DD CE B6E6        LDX      #RTETBL FROM READ TEST ERROR
B6E0 BD E7D6        JSR      DSCT    TABLE ARE DISPLAYED
B6E3 3E             BRPTEW WAI      WAIT (STOP)
B6E4 20 FD          BRA      BRPTEW

```

*

** END **

*

```

B6E6 RTETBL EQU *   READ TEST ERROR TABLE
B6E6 12             FCB      $12      "R"
B6E7 9F             FCB      $9F      "D."
B6E8 7A             FCB      $7A      "E"
B6E9 92             FCB      $92      "R."

```

*

```

B6EA 0003           RMB      3

```

*

```

B6ED BRWT EQU *    BUBBLE READ WRITE TEST

```

*

```

* - WRITES TWO PAGES AND READ A PAGE UNTIL BMS FULL
*   THEN READS TWO PAGES AND WRITES A PAGE UNTIL
*   BMS EMPTY
* - USES BWPT AND BRPT FOR WRITING AND READING PAGES

```

*

```

B6ED BD B41A        JSR      BMSE     EMPTY BMS
B6F0 BD B42B        JSR      BMSEE    ERASE BMS ERRORS
B6F3 CE 0000        LDX      #$0000
B6F6 FF 0950        STX      WPC      CLEAR READ AND WRITE
B6F9 FF 0956        STX      RPC      PAGE AND SEQUENCE
B6FC FF 0952        STX      WSC      COUNTERS
B6FF FF 0958        STX      RSC

```

*

```

*   WRITE UNTIL BMS FULL

```

*

```

B702 BD B5B5 BRWTW JSR      BWPT     WRITE A PAGE
B705 29 2B         BVS      BRWTR    SKIP IF BMS FULL
B707 FE 0950        LDX      WPC
B70A 08             INX              INC. WPC
B70B FF 0950        STX      WPC
B70E BD B5B5        JSR      BWPT     WRITE A PAGE
B711 29 1F         BVS      BRWTR    SKIP IF BMS FULL
B713 FE 0950        LDX      WPC
B716 08             INX              INC. WPC
B717 FF 0950        STX      WPC
B71A BD B68C        JSR      BRPT     READ A PAGE
B71D BD FFD0        JSR      CLDISP   DISPLAY RPC
B720 B6 0956        LDA      A  RPC
B723 F6 0957        LDA      B  RPC+1
B726 BD FFEB        JSR      WRITE4
B729 FE 0956        LDX      RPC
B72C 08             INX              INC. RPC
B72D FF 0956        STX      RPC
B730 20 D0          BRA      BRWTW    DO UNTIL BMS FULL

```

*

```

*   READ UNTIL BMS EMPTY

```



```

*
B732 86 08 BRWTR LDA A #$08 ADJUST SEQ. COUNT
B734 FE 0952 LDX WSC AS A LINE WAS NOT
B737 09 BRWTL3 DEX WRITTEN ON BMS FULL
B738 4A DEC A
B739 26 FC BNE BRWTL3
B73B FF 0952 STX WSC
B73E BD B68C BRWTRL JSR BRPT READ A PAGE
B741 29 37 BVS BSTX EXIT IF BMS EMPTY
B743 BD FFD0 JSR CLDISP DISPLAY RPC
B746 B6 0956 LDA A RPC
B749 F6 0957 LDA B RPC+1
B74C BD FFEB JSR WRITE4
B74F FE 0956 LDX RPC
B752 08 INX INC. RPC
B753 FF 0956 STX RPC
B756 BD B68C JSR BRPT READ A PAGE
B759 29 1F BVS BSTX EXIT IF BMS EMPTY
B75B BD FFD0 JSR CLDISP DISPLAY RPC
B75E B6 0956 LDA A RPC
B761 F6 0957 LDA B RPC+1
B764 BD FFEB JSR WRITE4
B767 FE 0956 LDX RPC
B76A 08 INX INC. RPC
B76B FF 0956 STX RPC
B76E BD B5B5 JSR BWPT WRITE A PAGE
B771 FE 0950 LDX WPC
B774 08 INX INC. WPC
B775 FF 0950 STX WPC
B778 20 C4 BRA BRWTRL DO UNTIL BMS EMPTY
*
B77A BSTX EQU * BMS TEST EXIT
*
B77A 96 5A BSTXL2 LDA A BMIBS WAIT UNTIL BMS INACT.
B77C 26 FC BNE BSTXL2 (WRITE NOT ACTIVE)
B77E 96 5B LDA A BMOBS
B780 26 F8 BNE BSTXL2 (READ NOT ACTIVE)
*
B782 CE 0940 LDX #FNEW LOC.OF BMS ERROR LIST
B785 86 FF LDA A #$FF VALUE IF NOT ERRORS
B787 A1 00 BSTXL1 CMP A 0,X CHECK FOR ANY ERROR
B789 26 17 BNE BSTXS1 SKIP IF ERROR
B78B 08 INX
B78C 8C 0950 CPX #RUCE+1 DO UNTIL LAST ERROR
B78F 26 F6 BNE BSTXL1 COUNTER CHECKED
*
B791 C6 03 LDA B #3 AFTER CHARACTER
B793 BD FFCD JSR DSPDP DISPLAY DECIMAL PIONT
B796 C6 04 LDA B #4
B798 86 04 LDA A #4
B79A CE E740 LDX #D104TB DISPLAY "10 4"
B79D BD E7D6 JSR DSCT
B7A0 20 0A BRA BSTW GO TO WAIT
*

```



```

B7A2 86 04    BSTXS1 LDA A  #4      4 CHARACTERS
B7A4 C6 04      LDA B  #4      STARTING AT LED 4
B7A6 CE B7AF    LDX      #DBSERT FROM BUB.SYST.ERR.TBL.
B7A9 BD E7D6    JSR      DSCT   ARE DISPLAYED("B.S.ER.")

```

*

```

B7AC 3E      BSTW   WAI      WAIT (STOP)
B7AD 20 FD    BRA    BSTW

```

*

** END **

*

```

B7AF DBSERT EQU *  DISPLAY BUBBLE SYSTEM ERROR TABLE
B7AF BE      FCB    $BE    "B."
B7B0 EE      FCB    $EE    "S."
B7B1 7A      FCB    $7A    "E"
B7B2 92      FCB    $92    "R."

```

*

*

*

* JUMP TABLE FOR ROM 7

*

```

B7DD      ORG    $B7DD

```

*

```

B7DD 7E B41A    JMP    BMSE
B7E0 7E B14E    JMP    BDIH
B7E3 7E B044    JMP    BCIH
B7E6 7E B175    JMP    BMSI
B7E9 7E B2BE    JMP    BMSO
B7EC 7E B4CA    JMP    TNDB
B7EF 7E B3E2    JMP    BMWF
B7F2 7E B43F    JMP    BMSD
B7F5 7E B4F6    JMP    BMSU
B7F8 7E B3B8    JMP    BMIBI
B7FB 7E B3C9    JMP    BMOBI
      END

```


ROM8

M68SAM is the property of Motorola Spd, Inc.
Copyright 1974 by Motorola Inc.

Motorola M6800 Cross Assembler, Release 1.1

NAM ROM8 SAMPLE PROCESSING, STORAGE, AND CONTROL

```

*
*
*
*   TIMER ADDRESSES   ( MC6840 )
*
7C01      CR2      EQU      $7C01
*
*
0080                      ORG      $0080
0080 0002  FILTPR RMB 2  PNTR TO SET OF FILTER COEFF.
0082 0002  COEFPR RMB 2  PNTR TO FILT.COEFFICIENT ARRAY
0084 0001  SPLC   RMB 1  THE SAMPLE COUNTER
0085 0001  SPLCRS RMB 1  THE RESET VALUE OF THAT COUNTER
0086 0001  SYST   RMB 1  THE SYSTEM STATUS REGISTER
*           $00 => NOT SAMPLING,TRANSMISSION NOT ACTIVE
*           $0F => NOT SAMPLING,TRANSMISSION ACTIVE
*           $F0 => SAMPLING,TRANSMISSION NOT ACTIVE
*           $FF => SAMPLING,TRANSMISSION ACTIVE
0087 0001  MLAC   RMB 1  .NE. $00 WHEN MAINLINE ACTIVE
0088 0001  MLST   RMB 1  .EQ. $FF WHEN MAIN TO BE STOPED
0089 0001  FTNFLG RMB 1  .NE. $00 WHEN A FUNCTION ACTIVE
008A 0001  SIGFLG RMB 1  .NE. $00 WHEN "SIG" IS ACTIVE
008B 0001  SELFLG RMB 1  .NE. $00 WHEN FILTER SELECTED
008C 0001  BSILF  RMB 1  OXXXXXX1 WHEN BSIL USED,BMSI IDLE
*           %1XXXXXXX WHEN BSIL USED,BMSI BUSY
*           %00000000 WHEN BSIL NOT USED AND
*                   BMSI IDLE
008D 0002  EPLFX1 RMB 2  X REGISTER STORAGE LOCATIONS
008F 0002  EPLFX2 RMB 2  USED BY ROUTINE "EPLF"
0091 0001  BSOLF  RMB 1  .EQ. $00 IFF BSIL NOT USED
0092 0002  TPVEC  RMB 2  VECTOR TO TRANS. PROCESS
*
*
*   OTHER VARIABLES USED BY ROM8
*
      0490      Y      EQU      $0490
      0400      S      EQU      $0400
*
*   COEFFICIENT ARRAYS :
*
      04B8      A0      EQU      $04B8
      04C8      A1      EQU      $04C8
      04D8      A2      EQU      $04D8
      04E8      B1      EQU      $04E8
      04F8      B2      EQU      $04F8
      0508      C0      EQU      $0508

```


0518	C1	EQU	\$0518	
0528	C2	EQU	\$0528	
0538	D1	EQU	\$0538	
0548	D2	EQU	\$0548	
0015	BUF1SR	EQU	\$0015	
001E	BUF2SR	EQU	\$001E	
0022	DC1FLG	EQU	\$0022	
0032	SPLVEC	EQU	\$0032	
0059	BSS	EQU	\$0059	- BUBBLE SYSTEM STATUS
0650	BSIL	EQU	\$0650	- BMS INPUT LINE
0690	BSOL	EQU	\$0690	- BMS OUTPUT LINE
0049	RXD	EQU	\$0049	- RECIEVED BYTE (ASCII)
0042	FLGC	EQU	\$0042	- FLAG CHAR.(1 BYTE HEX)

*

*

*

MONITOR GLOBAL VARIABLES AND FLAGS

*

0000	READEN	EQU	\$0000	- READ ENABLE
0004	ENTRY1	EQU	\$0004	- ENTRIES VIA ENTR FUNCTION
0005	ENTRY2	EQU	\$0005	
0006	ENTRY3	EQU	\$0006	
0007	ENTRY4	EQU	\$0007	
0008	ENTRY5	EQU	\$0008	
0009	ENTRY6	EQU	\$0009	
000A	ENTRY7	EQU	\$000A	
000B	ENTRY8	EQU	\$000B	
000D	IXRV	EQU	\$000D	- X REG.STORAGE, MAIN LINES ONLY
0010	MONFLG	EQU	\$0010	- INDICATES MAINLINE STOPED
0011	SECFLG	EQU	\$0011	- IND. 2ND KEYBOARD FTN. SELECTED
0012	LEDNO	EQU	\$0012	- THE # OF THE LED TO BE WRITTEN
				INTO NEXT, (FIRST LED = #0)

*

*

*

CONSTANTS USED BY ROM8

*

0080	FRFR	EQU	\$80	- DATA BLOCK FRAME FRONT
0081	FREN	EQU	\$81	- DATA BLOCK FRAME END
B000	RM7RST	EQU	\$B000	- ROM7 RESET VECTOR

*

*

*

SOME TABLES USED BY ROM8

*

F7A5	ENDTBL	EQU	\$F7A5
F423	ERRORT	EQU	\$F423
E740	D104TB	EQU	\$E740
F780	EOFTBL	EQU	\$F780
E75D	COPYON	EQU	\$E75D
E748	FORTON	EQU	\$E748
F78C	PTIPON	EQU	\$F78C
E738	DDATAL	EQU	\$E738
F798	PTIPOF	EQU	\$F798

*

E77F	FILT2	EQU	\$E77F
E793	FILT4	EQU	\$E793
E7A7	FILT8	EQU	\$E7A7

E724	FILTA	EQU	\$E724
E7BB	FILTBP	EQU	\$E7BB

*
*
*
*

SUBROUTINES USED BY ROM8

FFCA	DSPSN	EQU	\$FFCA
FFCD	DSPDP	EQU	\$FFCD
FFD0	CLDISP	EQU	\$FFD0
FFD6	DTSR2	EQU	\$FFD6
FFDF	DSPXAB	EQU	\$FFDF
FFC1	READ2	EQU	\$FFC1
FFBB	WRITE0	EQU	\$FFBB
FFC4	WRITE2	EQU	\$FFC4
FFE8	READ4	EQU	\$FFE8
FFEB	WRITE4	EQU	\$FFEB
FFF1	DELAY2	EQU	\$FFF1
FBF6	SCNDSR	EQU	\$FBF6
FFBE	FLSH	EQU	\$FFBE
F3F1	TMSR	EQU	\$F3F1
F7E3	TX1CHS	EQU	\$F7E3
F7F5	TX2CHS	EQU	\$F7F5
F3D6	SAMP	EQU	\$F3D6
F3E5	TMHB	EQU	\$F3E5
F3EB	CKSM	EQU	\$F3EB
F3EE	GTXB	EQU	\$F3EE
EFD0	FILT	EQU	\$EFD0
F3F4	MDTI	EQU	\$F3F4
B7EF	BMWF	EQU	\$B7EF
E7DF	CFOF	EQU	\$E7DF
E7D0	GTBT	EQU	\$E7D0
B7DD	BMSE	EQU	\$B7DD
E7D6	DSCT	EQU	\$E7D6
E7E5	FDID	EQU	\$E7E5
F7F2	CLBUF2	EQU	\$F7F2
B7E9	BMSO	EQU	\$B7E9
E7DC	SPTP	EQU	\$E7DC
B7F2	BMSD	EQU	\$B7F2
B7E6	BMSI	EQU	\$B7E6
B7FB	BMOBI	EQU	\$B7FB

*
*
*
*

LINKAGE TABLE ENTRIES FOR SWI ROUTINES

F400	IERR	EQU	\$F400
F402	ISTX1	EQU	\$F402
F404	ISTX2	EQU	\$F404
F406	ISABX	EQU	\$F406
F408	ILABX	EQU	\$F408
F40A	IABTX	EQU	\$F40A
F40C	IAITX	EQU	\$F40C
F40E	IRX2	EQU	\$F40E
F410	IPSHX	EQU	\$F410

*


```

*
*****
*
B800                      ORG      $B800
*****
*
*
*
*
      B800      RM8RST EQU      *          ROM 8  RESET
*
B800 7F 0042          CLR      FLGC
*
B803 CE 0080          LDX      #FILTPR      CLEAR ROM8 FLAGS
B806 6F 00      RM8RL1 CLR      0,X
B808 08          INX
B809 8C 0094          CPX      #TPVEC+2
B80C 26 F8          BNE      RM8RL1
*
B80E 7E B000          JMP      RM7RST      GO TO ROM 7 RESET
*
**  END  **
*
*
      B811      F2SU      EQU      *          FILTER #2  SET UP
*
*  - STUB TO BE ENTERED WITH MONITOR TO SET
*  FILTER COEFFICIENTS TO THOSE OF ANY FILTER
*  - BEFORE STUB IS RUN, THE LOCATION OF THE
*  FILTER COEFFICIENTS ARE PLACED IN ENTRY1 USING
*  THE ENTER FUNCTION
*
B811 C6 07          LDA  B  #$07          DO ALL 8 CHANNELS
B813 DE 04      F2SUL1 LDX      ENTRY1      TRANSFER FILTER
B815 BD B8A7          JSR          TFCF      COEFFICIENTS
B818 5A          DEC  B
B819 2A F8          BPL      F2SUL1
B81B 7E BCF0          JMP      SYIDWL      GO TO WAIT LOOP
**  END  **
*
*
B81E      SPLP2  EQU      *          SAMPLING PROCESS 2
*
*  - USED FOR TESTING, DOES NOT USE THE BMS
*  AS A BUFFER
*
B81E 0E          CLI          ALLOW INTERRUPTS
B81F BD F3D6          JSR      SAMP      OBTAIN SAMPLE BLOCK
B822 01          NOP          IN S (.9MSEC)
*
B823 BD EFD0          JSR      FILT      FILTER, OUTPUT Y(15 SEC)
*
B826 7A 0084          DEC      SPLC      DEC. SAMPLE COUNTER
B829 26 28          BNE      SPLPX1      IF NOT ZERO, EXIT

```



```

B82B 96 85      LDA A  SPLCRS RESET THE SAMPLE CNTR
B82D 97 84      STA A  SPLC
*
B82F CE 0490    LDX      #Y          MOVE DATA TO
B832 BD F3F4    JSR      MDTI        INPUT (.656 MSEC)
*
B835 BD F3EB    JSR      CKSM  APP.CHECK SUM(.241 MSEC.)
*
B838 BD F3E5    JSR      TMHB      ADD TIME,(3 HEX),
*
B83B 96 42      LDA A  FLGC          (.159 MSEC.)
B83D B7 0666    STA A  BSIL+22      ADD FLAG BYTE(1 HEX)
B840 7F 0042    CLR      FLGC
*
B843 CE 0651    LDX      #BSIL+1    GENERATE
B846 BD F3EE    JSR      GTXB      TX BLOCK (2.5 MSEC.)
*
B849 3F         SWI TX  THE TX BLOCK(3.4MSEC FOR SWI)
B84A 0000       FDB      0,ISTX2  +(2.1MSEC. FOR TX2)
B84C F404
*
B84E BD FFBE    JSR      FLSH      FLASH AN "S" ON DISP.
B851 6E         FCB      $6E,$06  ( 66 USEC )
B852 06
*
B853 3B         SPLPX1 RTI
*
**  END  **
*
*      B854      SSPL      EQU      *          START SAMPLING
*
*      - KEYBOARD ACTIVATED FUNCTION
*      - CLEARS FILTER DATA
*      - STARTS THE SAMPLING PROCESS BY
*      ENABLING THE TIMER INTERRUPT (CLK2 TIMER #1)
*
B854 96 89      LDA A  FTNFLG      CONTINUE IF A FTN.
B856 27 07      BEQ      SSPLS0    NOT ACTIVE
*
B858 CE F423    LDX      #ERRORT    DISPLAY "ERROR"
B85B BD FFD6    JSR      DTSR2
B85E 3B         RTI
*
B85F 96 11      SSPLS0 LDA A  SECFLG  CHECK FOR 2ND FTN.
B861 27 06      BEQ      SSPLS1
*
B863 BD FBF6    JSR      SCNDSR
B866 7E B8DA    JMP      HSPL
*
B869 96 86      SSPLS1 LDA A  SYST    ALREADY SAMPLING?
B86B 85 F0      BIT A  #$F0
B86D 26 37      BNE      SSPLX1    EXIT IF SO
*

```



```

B86F 96 59          LDA A   BSS          SKIP IF BMS FULL
B871 2A 07          BPL     SSPLS2
B873 CE BF20        LDX     #DBMSFL DISPLAY "B.M.S.FULL"
B876 BD FFD6        JSR     DTSR2
B879 3B             RTI
*
B87A 96 8B          SSPLS2 LDA A   SELFLG SKIP IF FILT.SEL.MADE
B87C 26 07          BNE     SSPLS3
B87E CE BF28        LDX     #DUSEL  DISPLAY "USE SEL" SO
B881 BD FFD6        JSR     DTSR2  USER MAKES SELECTION
B884 3B             RTI
*
B885 96 85          SSPLS3 LDA A   SPLCRS  RESET SAMPLE COUNTER
B887 97 84          STA A   SPLC
B889 CE 0400        LDX     #S          CLEAR FILTER DATA
B88C 6F 00          SSPLL1 CLR     0,X
B88E 08             INX
B88F 8C 04A6        CPX     #Y+22
B892 26 F8          BNE     SSPLL1
B894 7F 008C        CLR     BSILF      IND.BSIL NOT IN USE
*
B897 86 F0          LDA A   #$F0      FLAG THE NEXT SAMPLE
B899 97 42          STA A   FLGC      AS THE FIRST SAMPLE
B89B 86 D7          LDA A   #$D7      START SAMPLING,
B89D B7 7C01        STA A   CR2      ENABLE CLK #2 INT.
B8A0 D6 86          LDA B   SYST      IND.SAMPLE PROCESS
B8A2 CA F0          ORA B   #$F0      ACTIVE
B8A4 D7 86          STA B   SYST
B8A6 3B             SSPLX1 RTI        EXIT
**  END  **
*
*
B8A7 TFCF EQU * TRANSFER FILTER COEFFICIENTS
*
* ASSUMES      B = CHANNEL #
* ASSUMES      X = LOCATION OF 10 2 BYTE FILTER COEFF.
* TRANSFERS THE FILTER COEFFICIENTS TO THE APPROPRIATE
* ELEMENT IN THE 10 COEFFICIENT ARRAYS IN RAM
* CHANGES A,X,CCR,CLEAR B BIT 7
*
B8A7 58             ASL B           B=B*2=CANNEL # * 2
B8A8 DF 80          STX           FILTPR  STORE FILTER LOC.
B8AA CE 04B8        LDX           #A0
B8AD DF 82          STX           COEFPR  INITIALIZE COEF.PTR.
*
B8AF DE 80          TFCFL1 LDX     FILTPR
B8B1 A6 00          LDA A   0,X      TRANSFER FIRST BYTE
B8B3 08             INX
B8B4 DF 80          STX           FILTPR  STORE FILT.COEF.LOC.
B8B6 DE 82          LDX           COEFPR
*
B8B8 3F             SWI
B8B9 0000          FDB           0,ISABX  STORE A AT B+X
B8BB F406

```



```

*
B8BD DE 80      LDX      FILTPR
B8BF A6 00      LDA A    0,X
B8C1 08        INX
B8C2 DF 80      STX      FILTPR
B8C4 DE 82      LDX      COEFPR
B8C6 08        INX
*
B8C7 3F        SWI
B8C8 0000      FDB      0,ISABX   STORE A AT B+X
B8CA F406
*
B8CC 3F        SWI                      ADD OFFSET TO
B8CD 000F      FDB      $000F,IAITX  NEXT ARRAY
B8CF F40C
*
B8D1 DF 82      STX      COEFPR
B8D3 8C 0558    CPX      #D2+16      ALL COEF. DONE?
B8D6 26 D7      BNE      TFCFL1      LOOP IF NOT DONE
*
B8D8 54        LSR B                  RESTORE B
B8D9 39        RTS
*
**  END    **
*
*
B8DA HSPL      EQU *  HALT SAMPLING
*
* - THIS ROUTINE STOPS THE SAMPLING PROCESS
* - SAMPLING CAN BE CONTINUED WITH THE SSPL FUNCTION
* - THIS IS A KEYBOARD ACTIVATED FUNCTION
*
B8DA 96 86      LDA A    SYST
B8DC 85 F0      BIT A    #$F0      EXIT
B8DE 27 09      BEQ      HSPLX1    IF NOT SAMPLING
*
B8E0 84 0F      AND A    #$0F
B8E2 97 86      STA A    SYST      IND.NOT SAMPLING
B8E4 86 97      LDA A    #$97      DISABLE CLK2,TIMER #1
B8E6 B7 7C01    STA A    CR2      INTERRUPT
B8E9 3B        HSPLX1 RTI
*
**  END    **
*
*
B8EA END      EQU *  END
*
* - A KEYBOARD ACTIVATED FUNCTION
* - MAY STOP THE SAMPLING PROCESS
* - INDICATES TO THE MAINLINE THAT THE DATA SENDING
*   PROCESS IS TO BE STOPED AFTER THE BMS IS EMPTY
*
B8EA 96 86      LDA A    SYST      CONTINUE IF
B8EC 85 0F      BIT A    #$0F      DATA IS BEING SENT

```



```

B8EE 26 07          BNE      ENDS1
B8F0 CE BF30        LDX      #DTTXSA  DISPLAY "TRY TX.SA."
B8F3 BD FFD6        JSR      DTSR2    TO IND."TX" OR "SEND"
B8F6 3B             RTI             SHOULD BE USED
*
B8F7 7F 0086 ENDS1  CLR      SYST IND.MAIN TO STOP ON BMS MT
B8FA 86 97          LDA A    #$97  DISABLE CLK2,TIMER #2
B8FC B7 7C01        STA A    CR2    INTERRUPT
B8FF 7C 0089        INC      FTNFLG  IND. FUNCTION ACTIVE
B902 0E             CLI             ALLOW FOR BMS INT.
B903 BD B7EF        JSR      BMWF    WRITE LAST PG TO BMS
*
B906 BD FFD0        JSR      CLDISP
B909 CE F7A5        LDX      #ENDTBL
B90C 86 01          LDA A    #$01    DISPLAY "E"
B90E 5F             CLR B
B90F BD E7D6        JSR      DSCT
B912 7F 0089        CLR      FTNFLG
*
B915 3B             ENDX1  RTI
**  END  **
*
*
B916      SIG      EQU      *          SIGNON
*
*  - A KEYBOARD ACTIVATED FUCTION USED TO
*  SIGNON TO THE MTS SYSTEM
*
B916 96 86          LDA A    SYST      SKIP IF
B918 27 07          BEQ      SIGS1     SYSTEM APPEARS IDLE
B91A CE F423 SIGXE  LDX      #ERRORT  DISPLAY "ERROR" AS
B91D BD FFD6        JSR      DTSR2    SYSTEM MUST BE IDLE
B920 3B             RTI
*
B921 96 87          SIGS1  LDA A    MLAC      SKIF IF
B923 27 07          BEQ      SIGS2     MAINLINE INACTIVE
B925 CE BF50        LDX      #DTRLTR  DISPLAY "TRY.LATER"
B928 BD FFD6        JSR      DTSR2    AS MAIN STILL ACTIVE
B92B 3B             RTI
*
B92C 96 8A          SIGS2  LDA A    SIGFLG   SKIP IF
B92E 27 04          BEQ      SIGS3     "SIG" NOT ACTIVE
B930 7F 008A        CLR      SIGFLG   INDICATE "SIG" TO STOP
B933 3B             RTI
*
B934 96 89          SIGS3  LDA A    FTNFLG   EXIT
B936 26 E2          BNE      SIGXE     IF ANOTHER FTN ACT.
B938 7C 008A        INC      SIGFLG  IND."SIG" AND FTN ACT.
B93B 7C 0089        INC      FTNFLG
B93E BD FFD0        JSR      CLDISP
B941 CE B9F0        LDX      #DSIGON  DISPLAY "SIG"
B944 86 03          LDA A    #$03
B946 5F             CLR B
B947 BD E7D6        JSR      DSCT

```



```

B94A 0E          CLI          ALLOW FOR TX INT.
*
B94B 7F 0049 SIGL2 CLR      RXD
B94E CE B9F8     LDX      #LTTBL  CHECK IF SIGNED ON,
B951 BD F7E3     JSR      TX1CHS   SEND "$LIST T"
B954 C6 16       LDA B    #$16     WAIT 8 SEC. MAX.
*
B956 BD FFF1 SIGL1 JSR      DELAY2
B959 00F9        FDB      $00F9    WAIT .5 SEC
B95B 96 49       LDA A    RXD
B95D 81 5D       CMP A    #$5D     IF "]" WAS RXED,
B95F 27 27       BEQ      SIGX1    THEN SIGNON OK,EXIT
B961 4D         TST A          IF A CHAR.RXED,THEN
B962 26 36       BNE      SIGS4    MODEM UP,NOT SIGNED ON
B964 7D 008A     TST      SIGFLG
B967 27 15       BEQ      SIGX2    EXIT IF"SIG"IS TO STOP
B969 5A         DEC B
B96A 26 EA       BNE      SIGL1    LOOP UNTIL 8 SEC. UP
*
B96C CE BA0B     LDX      #MMBR    SEND "MODEM MUST BE
B96F BD F7E3     JSR      TX1CHS   READY' BEL' "
B972 BD FFF1     JSR      DELAY2
B975 00F9        FDB      $00F9    WAIT .5 SEC
B977 7D 008A     TST      SIGFLG
B97A 27 02       BEQ      SIGX2    EXIT IF"SIG"TO STOP
B97C 20 CD       BRA      SIGL2    DO UNTIL MODEM UP
*
B97E CE BA03 SIGX2 LDX      #DSIGFT  DISPLAY "SIG. FAIL"
B981 BD FFD6     JSR      DTSR2
B984 7F 0089     CLR      FTNFLG    IND.FTN NOT ACTIVE
B987 3B         RTI
*
B988 CE B9F0 SIGX1 LDX      #DSIGON  DISPLAY "SIGNED.ON"
B98B BD FFD6     JSR      DTSR2
B98E 7F 008A     CLR      SIGFLG    IND."SIG" AND FTN
B991 7F 0089     CLR      FTNFLG    NOT ACTIVE
B994 3B         RTI
*
B995 0003        RMB      3
B998 20 B1 SIGL2B BRA      SIGL2
*
B99A 7F 0049 SIGS4 CLR      RXD
B99D BD FFF1     JSR      DELAY2    WAIT .5 SEC.
B9A0 00F9        FDB      $00F9
B9A2 96 49       LDA A    RXD      WAIT UNTIL
B9A4 26 F4       BNE      SIGS4    NO CHAR. RXED
*
B9A6 CE BF76     LDX      #SIGID    SEND "SIG IDID"
B9A9 BD F7E3     JSR      TX1CHS
B9AC 7D 008A SIGL4 TST      SIGFLG
B9AF 27 CD       BEQ      SIGX2    EXIT IF"SIG"TO STOP
B9B1 96 49       LDA A    RXD
B9B3 81 3F       CMP A    #$3F     WAIT FOR FIRST "?"
B9B5 26 F5       BNE      SIGL4

```



```

B9B7 7F 0049      CLR      RXD
B9BA 7D 008A SIGL5 TST      SIGFLG
B9BD 27 BF        BEQ      SIGX2      EXIT IF "SIG" TO STOP
B9BF 96 49        LDA A    RXD
B9C1 81 3F        CMP A    #$3F      WAIT FOR NEXT "?"
B9C3 26 F5        BNE      SIGL5
*
B9C5 7F 0049 SIGL7 CLR      RXD
B9C8 CE BF82      LDX      #PASSWD    SEND PASSWORD
B9CB BD F7E3      JSR      TX1CHS
B9CE 7D 008A SIGL6 TST      SIGFLG
B9D1 27 AB        BEQ      SIGX2      EXIT IF "SIG" TO STOP
B9D3 96 49        LDA A    RXD
B9D5 27 F7        BEQ      SIGL6      WAIT FOR NEXT CHAR.
*
B9D7 BD FFF1 SIGL8 JSR      DELAY2
B9DA 01F2          FDB      $01F2      DELAY 1 SEC.
B9DC 96 49        LDA A    RXD
B9DE 81 3F        CMP A    #$3F      IF A "?" RXED,
B9E0 27 E3        BEQ      SIGL7      THEN RETX PASSWORD
B9E2 7D 008A      TST      SIGFLG
B9E5 27 97        BEQ      SIGX2      EXIT IF "SIG" TO STOP
B9E7 7F 0049      CLR      RXD
B9EA 4D           TST A
B9EB 26 EA        BNE      SIGL8      DO UNTIL
B9ED 7E B94B      JMP      SIGL2      NO CHAR. RXED
                                     TEST IF SIGNED ON
*
**      END      **
*
*
*
B9F0 DSIGON EQU *      DISPLAY SIGNED ON
*
B9F0 6E           FCB      $6E      "S"
B9F1 04           FCB      $04      "I"
B9F2 7C           FCB      $7C      "G"
B9F3 16           FCB      $16      "N"
B9F4 7A           FCB      $7A      "E"
B9F5 9F           FCB      $9F      "D."
B9F6 1E           FCB      $1E      "O"
B9F7 16           FCB      $16      "N"
*
*
B9F8 LTTBL EQU *      LIST T
*
B9F8 24           FCC      '$LIST T'
B9F9 4C
B9FA 49
B9FB 53
B9FC 54
B9FD 20
B9FE 54
B9FF 0D           FCB      $0D,$0A,$7F,$04
BA00 0A

```


BA01 7F

BA02 04

*

*

BA03 DSIGFT EQU * DISPLAY SIGNON FAIL

*

BA03 6E FCB \$6E "S"

BA04 04 FCB \$04 "I"

BA05 FC FCB \$FC "G"

BA06 00 FCB \$00 " "

BA07 72 FCB \$72 "F"

BA08 77 FCB \$77 "A"

BA09 04 FCB \$04 "I"

BA0A 38 FCB \$38 "L"

*

*

BA0B MMBR EQU *

*

BA0B 07 FCB \$07 "BEL"

BA0C 20 FCC ' ** MODEM MUST BE "READY" **'

BA0D 2A

BA0E 2A

BA0F 20

BA10 20

BA11 4D

BA12 4F

BA13 44

BA14 45

BA15 4D

BA16 20

BA17 20

BA18 4D

BA19 55

BA1A 53

BA1B 54

BA1C 20

BA1D 20

BA1E 42

BA1F 45

BA20 20

BA21 22

BA22 52

BA23 45

BA24 41

BA25 44

BA26 59

BA27 22

BA28 20

BA29 2A

BA2A 2A

BA2B 07 FCB \$07 "BEL"

BA2C 0D FCB \$0D, \$0A, \$7F, \$04

BA2D 0A

BA2E 7F

BA2F 04

*

BA30 CLBS EQU * CLEAR BUBBLE SYSTEM

*

- * - A KEYBOARD ACTIVATED FUNCTION
- * - ALLOWS THE BUBBLE SYSTEM TO BE EMPTIED FROM
- * THE KEYBOARD
- * - AFTER THIS ROUTINE IS RUN, "SAVE" SOULD BE RUN
- * BEFORE POWERDOWN TO SAVE THE NEW STATUS OF THE
- * BMS CONTENTS

*

BA30	96	86		LDA	A	SYST	EXIT
BA32	27	07		BEQ		CLBSS1	IF SYSTEM NOT IDLE
BA34	CE	F423	CLBSXE	LDX		#ERRORT	DISPLAY "ERROR" AS
BA37	BD	FFD6		JSR		DTSR2	SYSTEM MUST BE IDLE
BA3A	3B			RTI			

*

BA3B	96	87	CLBSS1	LDA	A	MLAC	SKIP IF
BA3D	27	07		BEQ		CLBSS2	MAINLINE NOT ACTIVE
BA3F	CE	BF50		LDX		#DTRLTR	DISPLAY "TRY.LATER" AS
BA42	BD	FFD6		JSR		DTSR2	MAINLINE STILL ACT.
BA45	3B			RTI			

*

BA46	96	89	CLBSS2	LDA	A	FTNFLG	EXIT IF A
BA48	26	EA		BNE		CLBSXE	FUNCTION ACTIVE
BA4A	7C	0089		INC		FTNFLG	IND. FUNCTION ACTIVE
BA4D	BD	FFD0		JSR		CLDISP	
BA50	CE	BA83		LDX		#DMTBS	DISPLAY "MT.BS?"
BA53	86	06		LDA	A	#\$06	
BA55	5F			CLR	B		
BA56	BD	E7D6		JSR		DSCT	
BA59	0E			CLI			ALLOW FOR TX INT.
BA5A	CE	BA89		LDX		#BSMT	SEND "BUBBLE SYSTEM
BA5D	BD	F7E3		JSR		TX1CHS	TO BE EMPTIED??'BEL' "

*

BA60	BD	E7D0		JSR		GTBT	GET CONFIRMATION
BA63	25	11		BCS		CLBSXA	EXIT IF NONE
BA65	81	A1		CMP	A	#\$A1	
BA67	26	0D		BNE		CLBSXA	EXIT IF NOT CONFIRMED

*

BA69	BD	B7DD		JSR		BMSE	EMPTY BMS
BA6C	CE	BF48		LDX		#DBSMT	DISPLAY "B.S. MT."
BA6F	BD	FFD6		JSR		DTSR2	
BA72	7F	0089		CLR		FTNFLG	IND. FTN NOT ACTIVE
BA75	3B			RTI			

*

BA76	BD	FFD0	CLBSXA	JSR		CLDISP	"CLBS" ABORT EXIT
BA79	7F	0089		CLR		FTNFLG	IND. FTN NOT ACTIVE
BA7C	3B			RTI			

*

** END **

BA7D 0006 RMB 6

*

*

BA83 DMTBS EQU * DISPLAY "EMPTY BUBBLE SYSTEM?"

*

BA83	16	FCB	\$16	"M"
BA84	06	FCB	\$06	
BA85	C5	FCB	\$C5	"T."
BA86	BE	FCB	\$BE	"B."
BA87	EE	FCB	\$EE	"S."
BA88	53	FCB	\$53	"?"

*

*

BA89 BSMT EQU * "'BEL' BUBBLE SYSTEM TO BE EMPTIED' BEL' "

*

BA89	07	FCB	\$07	"BEL"
BA8A	42	FCC	'BUBBLE SYSTEM TO BE EMPTIED ??'	
BA8B	55			
BA8C	42			
BA8D	42			
BA8E	4C			
BA8F	45			
BA90	20			
BA91	53			
BA92	59			
BA93	53			
BA94	54			
BA95	45			
BA96	4D			
BA97	20			
BA98	54			
BA99	4F			
BA9A	20			
BA9B	42			
BA9C	45			
BA9D	20			
BA9E	45			
BA9F	4D			
BAA0	50			
BAA1	54			
BAA2	49			
BAA3	45			
BAA4	44			
BAA5	20			
BAA6	3F			
BAA7	3F			
BAA8	07	FCB	\$07	"BEL"
BAA9	0D	FCB	\$0D,\$0A,\$7F,\$04 "CR","LF","DEL",ENDOFSTRING	
BAAA	0A			
BAAB	7F			
BAAC	04			

*

*

*

BAAD SEL EQU * SELECT

*

* - A KEYBOARD ACTIVATED FUNCTION


```

*      - USED TO GET A ONE BYTE CODE FROM THE USER
*      TO SELECT A TRANSMISSION PROCESS, A SAMPLE
*      PROCESS, A SAMPLE FREQUENCY, AND A
*      SET OF FILTER COEFFICIENTS
*
BAAD 96 11          LDA A  SECFLG
BAAF 27 06          BEQ    SELS1  SKIP IF NOT 2ND FTN
*
BAB1 BD FBF6        JSR      SCNDSR
BAB4 7E BA30        JMP      CLBS   SECOND FTN.  SELECTED
*
BAB7 96 86  SELS1  LDA A  SYST      EXIT
BAB9 27 07        BEQ    SELS2  IF SYSTEM NOT IDLE
BABB CE F423  SELXE LDX    #ERRORT  DISPLAY "ERROR"
BABE BD FFD6        JSR      DTSR2
BAC1 3B          RTI
*
BAC2 96 87  SELS2  LDA A  MLAC      CONTINUE IF
BAC4 27 07        BEQ    SELS3  MAINLINE NOT ACTIVE
BAC6 CE BF50      LDX    #DTRLTR  DISPLAY"TRY.LATER"AS
BAC9 BD FFD6        JSR      DTSR2  MAINLINE STILL ACTIVE
BACC 3B          RTI
*
BACD 96 89  SELS3  LDA A  FTNFLG
BACF 26 EA        BNE    SELXE  EXIT IF FTN.  ACTIVE
*
BAD1 7C 0089      INC      FTNFLG  IND. A FTN.  ACTIVE
BAD4 96 8B        LDA A  SELFLG  SKIP IF
BAD6 27 15        BEQ    SELS4  SEL.NOT YET MADE
*
BAD8 BD FFD0      JSR      CLDISP  DISPLAY "RESEL?" TO
BADB CE BBCD      LDX      #DRESEL  TO ASK FOR
BADE 86 06        LDA A  #$06      RESEL.  CONFIRMATION
BAE0 5F          CLR B
BAE1 BD E7D6      JSR      DSCT
BAE4 BD E7D0      JSR      GTBT    GET A BYTE FROM USER
BAE7 25 79        BCS      SELX1  EXIT IF DISPLAY CLEAR
BAE9 81 A1        CMP A  #$A1
BAEB 26 72        BNE      SELX2  EXIT IF NOT CONFIRMED
*
BAED BD FFD0  SELS4 JSR      CLDISP
BAF0 CE BBCF      LDX      #DRESEL+2  DISPLAY "SEL."
BAF3 86 03        LDA A  #$03
BAF5 5F          CLR B
BAF6 BD E7D6      JSR      DSCT
BAF9 BD E7D0      JSR      GTBT    GET SELECTION CODE
BAFC 25 64        BCS      SELX1  EXIT IF DISPLAY CLEAR
*
BAFE 85 10        BIT A  #$10  IF THE FIRST NIB.IS 1,
BB00 27 0B        BEQ      SELS6  THE FORTRAN PROCESS WAS
BB02 85 E0        BIT A  #$E0      SELECTED
BB04 26 07        BNE      SELS6
BB06 84 0F        AND A  #$0F      CLEAR FIRST NIBBLE
BB08 CE E748      LDX      #FORTON

```



```

BB0B 20 03          BRA      SELS7      OTHERWISE
BB0D CE E75D SELS6  LDX      #COPYON   THE COPY PROCESS
BB10 DF 92          STX      TPVEC      WAS SELECTED
*
BB12 CE BB6C          LDX      #SELTBL   GET SELECTION TABLE
BB15 C6 06          LDA      B $06      SET # BYTES/ENTRY
BB17 BD E7E5          JSR      FDID      FIND LOC.OF SEL.CODE
BB1A 24 0D          BCC      SELS5      SKIP IF CODE FOUND
BB1C CE F423          LDX      #ERRORT   DISPLAY "ERROR"
BB1F BD FFD6          JSR      DTSR2
BB22 7F 0089          CLR      FTNFLG    IND.FTN. NOT ACTIVE
BB25 7F 008B          CLR      SELFLG    IND. NO SEL. MADE
BB28 3B              RTI
*
BB29 E6 01          SELS5  LDA      B 1,X    SET SAMPLE CNTR RESET
BB2B D7 85          STA      B SPLCRS
BB2D DF 0A          STX      ENTRY7
BB2F EE 04          LDX      4,X          SET VECTOR TO
BB31 DF 32          STX      SPLVEC      SAMPLE ROUTINE
BB33 DE 0A          LDX      ENTRY7
BB35 EE 02          LDX      2,X          SAVE LOC. FILT.COEF.
BB37 DF 0A          STX      ENTRY7
BB39 C6 07          LDA      B $07        STARTING WITH CH. 7
BB3B DE 0A          SELL1  LDX      ENTRY7
BB3D BD B8A7          JSR      TFCF        SET COEFF. FOR A CH.
BB40 5A              DEC      B
BB41 2A F8          BPL      SELL1        DO FOR ALL CHANNELS
*
BB43 BD FFD0          JSR      CLDISP
BB46 CE BBCF          LDX      #DRESEL+2  DISPLAY "SEL."
BB49 86 03          LDA      A $03
BB4B 5F              CLR      B
BB4C BD E7D6          JSR      DSCT
BB4F CE E740          LDX      #D104TB    AND DISPLAY "10 4"
BB52 86 04          LDA      A $04
BB54 C6 04          LDA      B $04
BB56 BD E7D6          JSR      DSCT
BB59 C6 0F          LDA      B $0F        INDICATE A SEL. MADE
BB5B D7 8B          STA      B SELFLG
BB5D 20 03          BRA      SELX1
*
BB5F BD FFD0 SELX2  JSR      CLDISP
BB62 7F 0089 SELX1  CLR      FTNFLG    IND. FTN NOT ACTIVE
BB65 3B              RTI
*
**      END      **
BB66 0006          RMB      6
*
*
BB6C SELTBL EQU      *      SELECTION TABLE
*
* - A TABLE OF ONE BYTE SELECTION CODES AND
*   THEIR ASSOCIATED RESAMPLE FREQUENCIES,
*   FILTER COEFFICIENTS, AND SAMPLE PROCESSES

```



```

* - TABLE FORMAT:   ( 6 BYTES / ENTRY )
*                   FCB   (ONE HEX BYTE SELECTION CODE)
*                   FCB   (RESAMPLE COUNT => FREQUENCY)
*                   FDB   (LOCATION OF FILTER COEFFICIENTS)
*                   FDB   (LOCATION OF SAMPLE PROCESS ROUTINE)
* - THE RESAMPLE COUNT ASSUMES AN INITIAL
*   SAMPLE FREQUENCY OF 40 HZ.
*
BB6C 02          FCB   $02
BB6D 14          FCB   20          2 HZ. RESAMPLE
BB6E E77F        FDB   FILT2       .6 HZ. CUT OFF
BB70 BFD0        FDB   SPP1V      DEFAULT SAMPLE PROCESS
*
BB72 04          FCB   $04
BB73 0A          FCB   10          4 HZ. RESAMPLE
BB74 E793        FDB   FILT4       1.2 HZ. CUT OFF
BB76 BFD0        FDB   SPP1V      DEFAULT SAMPLE PROCESS
*
BB78 08          FCB   $08
BB79 05          FCB   5           8 HZ. RESAMPLE
BB7A E7A7        FDB   FILT8       2.4 HZ. CUT OFF
BB7C BFD0        FDB   SPP1V      DEFAULT SAMPLE PROCESS
*
BB7E 0A          FCB   $0A
BB7F 04          FCB   4           10 HZ. RESAMPLE
BB80 E724        FDB   FILTA       3.1 HZ. CUT OFF
BB82 BFD0        FDB   SPP1V      DEFAULT SAMPLE PROCESS
*
BB84 B2          FCB   $B2
BB85 14          FCB   20          2 HZ. RESAMPLE
BB86 E7BB        FDB   FILTBP      BYPASS FILTER
BB88 BFD0        FDB   SPP1V      DEFAULT SAMPLE PROCESS
*
BB8A B4          FCB   $B4
BB8B 0A          FCB   10          4 HZ. RESAMPLE
BB8C E7BB        FDB   FILTBP      BYPASS FILTER
BB8E BFD0        FDB   SPP1V      DEFAULT SAMPLE PROCESS
*
BB90 B8          FCB   $B8
BB91 05          FCB   05          8 HZ. RESAMPLE
BB92 E7BB        FDB   FILTBP      BYPASS FILTER
BB94 BFD0        FDB   SPP1V      DEFAULT SAMPLE PROCESS
*
BB96 BA          FCB   $BA
BB97 04          FCB   4           10 HZ. RESAMPLE
BB98 E7BB        FDB   FILTBP      BYPASS FILTER
BB9A BFD0        FDB   SPP1V      DEFAULT SAMPLE PROCESS
*
BB9C E2          FCB   $E2
BB9D 14          FCB   20          2 HZ. RESAMPLE
BB9E E77F        FDB   FILT2       .6 HZ. CUTOFF
BBA0 BFD3        FDB   SPLP2V     BMS BY PASS
*
BBA2 E4          FCB   $E4

```


BBA3	0A	FCB	10	4 HZ. RESAMPLE
BBA4	E793	FDB	FILT4	1.2 HZ. CUTOFF
BBA6	BFD3	FDB	SPLP2V	BMS BY PASS
*				
BBA8	E8	FCB	\$E8	
BBA9	05	FCB	05	8 HZ. RESAMPLE
BBAA	E7A7	FDB	FILT8	2.4 HZ. CUT OFF
BBAC	BFD3	FDB	SPLP2V	BMS BY PASS
*				
BBAE	EA	FCB	\$EA	
BBAF	04	FCB	4	10 HZ. RESAMPLE
BBB0	E724	FDB	FILTA	3.1 HZ. CUT OFF
BBB2	BFD3	FDB	SPLP2V	BMS BY PASS
*				
BBB4	F2	FCB	\$F2	
BBB5	14	FCB	20	2 HZ. RESAMPLE
BBB6	E7BB	FDB	FILTBP	BYPASS FILTER
BBB8	BFD3	FDB	SPLP2V	BMS BY PASS
*				
BBBA	F4	FCB	\$F4	
BBBB	0A	FCB	10	4 HZ. RESAMPLE
BBBC	E7BB	FDB	FILTBP	BYPASS FILTER
BBBE	BFD3	FDB	SPLP2V	BMS BY PASS
*				
BBC0	F8	FCB	\$F8	
BBC1	05	FCB	05	8 HZ. RESAMPLE
BBC2	E7BB	FDB	FILTBP	BYPASS FILTER
BBC4	BFD3	FDB	SPLP2V	BMS BY PASS
*				
BBC6	FA	FCB	\$FA	
BBC7	04	FCB	4	10 HZ. RESAMPLE
BBC8	E7BB	FDB	FILTBP	BYPASS FILTER
BBCA	BFD3	FDB	SPLP2V	BMS BY PASS
*				
BBCC	FF	FCB	\$FF	" END OF TABLE "
*				
*				
*				
*				
BBCD	DRESEL EQU	*	DISPLAY	"RESELECT?"
*				
BBCD	12	FCB	\$12	"R"
BBCE	7A	FCB	\$7A	"E"
BBCF	6E	FCB	\$6E	"S"
BBD0	7A	FCB	\$7A	"E"
BBD1	B8	FCB	\$B8	"L."
BBD2	53	FCB	\$53	"?"
*				
*				
BBD3	TX	EQU	*	TRANSMIT
*				
* - A KEYBOARD ACTIVATED FUNCTION TO START				
* THE MAIN LINE PROCESS WHICH READS DATA FROM				
* THE BMS AND TRANSMITS IT TO THE MTS SYSTEM				


```

*
BBD3 96 89      LDA A   FTNFLG   EXIT IF
BBD5 26 2A      BNE     TXXE     A FUNCTION ACTIVE
BBD7 96 11      LDA A   SECFLG   CHECK IF 2ND FTN
BBD9 27 06      BEQ     TXS0     SELECTED
*
BBDB BD FBF6    JSR     SCNDSR
BBDE 7E BDCB    JMP     NOTX     GO TO 2ND FUNCTION
*
BBE1 96 8B      TXS0   LDA A   SELFLG   EXIT IF TX. PROCESS
BBE3 26 07      BNE     TXS5     WAS NOT SELECTED
BBE5 CE BF28    LDX     #DUSEL   DISPLAY "USE SEL"
BBE8 BD FFD6    JSR     DTSR2
BBEB 3B        RTI
*
BBEC 96 86      TXS5   LDA A   SYST     SKIP IF
BBEE 85 0F      BIT A   #$0F     THE SYSTEM IS NOT
BBF0 27 16      BEQ     TXS3     ALREADY SENDING DATA
BBF2 85 F0      BIT A   #$F0
BBF4 26 0B      BNE     TXXE     EXIT IF SAMPLING
BBF6 D6 59      LDA B   BSS
BBF8 26 07      BNE     TXXE     EXIT IF BMS NOT EMPTY
BBFA CE BF48    LDX     #DBSMT   DISPLAY "B.S. MT."
BBFD BD FFD6    JSR     DTSR2
BC00 3B        RTI
*
BC01 CE F423    TXXE   LDX     #ERRORT  DISPLAY "ERROR"
BC04 BD FFD6    JSR     DTSR2
BC07 3B        RTI
*
BC08 96 87      TXS3   LDA A   MLAC     SKIP IT MAIN
BC0A 27 07      BEQ     TXS4     LINE NOT ACTIVE
BC0C CE BF50    LDX     #DTRLTR   DISPLAY "TRY.LATER"
BC0F BD FFD6    JSR     DTSR2     AS MAINLINE IS BUSY
BC12 3B        RTI             ( "END" WAS RUN )
*
BC13 96 86      TXS4   LDA A   SYST     IND. TRANSMISSION
BC15 8A 0F      ORA A   #$0F     IS TO START
BC17 97 86      STA A   SYST
BC19 3B        RTI
*
**   END   **
*
*
BC1A      SDBM EQU * SEND DATA FROM BUBBLE, MAINLINE
*
* - THE MAIN LINE ROUTINE WHICH READS LINES FROM
*   THE BMS AND TRANSMITS THEM TO THE MTS SYSTEM
* - AN INFINITE LOOP WHICH EXITS TO A WAIT LOOP
*   ( THE SYSTEM IDLE STATE ) ON THE FOLLOWING
*   CONDITIONS
*       1) BMS IS EMPTY AND SYST IS CLEAR
*       2) WHEN "MLST"=$FF
*       3) IF IT TAKES TOO LONG TO TRANSMIT A

```



```

*           LINE TO THE MTS SYSTEM ( BUF2 DOES
*           NOT EMPTY )
*       4) IF THERE ARE 3 CONSECUTIVE FRAMING
*           ERRORS IN READING A LINE FROM THE BMS
* - THIS IS THE LOWEST PRIORITY PROCESS IN THE SYSTEM
* AND IS INTERRUPTED BY ALL OTHERS
*
* MAINLINE LOOP ENTRY LOCATION
*
BC1A 86 FF          LDA A  #$FF          IND. MAINLINE ACTIVE
BC1C 97 87          STA A  MLAC
BC1E 7F 0088        CLR          MLST          CLEAR MAIN STOP REQ.
BC21 97 22          STA A  DC1FLG         IND. READY TO TX
BC23 0E             CLI              ALLOW FOR INTERRUPTS
*
BC24 BD BD78        JSR          CFON      START COPY/FORT IN MTS
BC27 CE BF38        LDX          #DSNDG   DISPLAY "SENDING"
BC2A BD FFD6        JSR          DTSR2
*
BC2D SDBML EQU *    BEGINNING OF THE MAIN LINE LOOP
*
BC2D 96 91          LDA A  BSOLF         SKIP IF THE LINE IN
BC2F 26 19          BNE          SDBMS4   BSOL IS NOT TXED
*
BC31 CE 0690        LDX          #BSOL     LOCATION OF LINE
BC34 C6 18          LDA B  #24           # OF BYTES IN LINE
BC36 BD B7E9        JSR          BMSO      GET A LINE FORM BMS
BC39 28 0D          BVC          SDBMS1    SKIP IF BMS NOT MT
*
BC3B D6 86          SDBML1 LDA B  SYST     EXIT IF SYST CLEAR
BC3D 26 03          BNE          SDBMS0
BC3F 7E BCDF        JMP          SDBMX1
BC42 D6 59          SDBMS0 LDA B  BSS      WAIT UNTIL
BC44 27 F5          BEQ          SDBML1    BMS NOT EMPTY
BC46 20 E5          BRA          SDBML     THEN TRY TO GET
*                                     ANOTHER LINE
BC48 27 E3          SDBMS1 BEQ          SDBML RETRY IF NOT RECIEVED
*
BC4A 86 0F          SDBMS4 LDA A  #$0F     IND. BSOL USED
BC4C 97 91          STA A  BSOLF
BC4E BD BD25        JSR          EPLF      PROPER LINE FRAMING?
BC51 29 E8          BVS          SDBML1    WAIT IF BMS EMPTY
BC53 26 03          BNE          SDBMSA    EXIT IF LINE NOT
BC55 7E BCBC        JMP          SDBMX2    FRAMED
*
BC58 C6 05          SDBMSA LDA B  #$05
BC5A 37             PSH B                *** SAVE # OF TIME OUTS
BC5B 34             SDBML3 DES *HG*
BC5C 34             DES *MG*
BC5D 34             DES *SG*
BC5E 0F             SEI
BC5F BD F3F1        JSR          TMSR      GET CURRENT TIME
BC62 0E             CLI
*

```



```

BC63 96 88      SDBML2 LDA A  MLST
BC65 81 FF      CMP A  #$FF      EXIT IF MAIN LINE
BC67 27 06      BEQ      SDBMXA    IS TO BE STOPED
BC69 96 22      LDA A  DC1FLG    SKIP IF WE CAN
BC6B 26 22      BNE      SDBMS2    TRANSMIT
BC6D 20 06      BRA      SDBMS3    SKIP EXIT
BC6F 31          SDBMXA INS      *SG*
BC70 31          INS      *MG*
BC71 31          INS      *HG*
BC72 31          INS      ***
BC73 20 51      BRA      SDBMX3
BC75 86 01      SDBMS3 LDA A  #$01    CHECK FOR 1 MIN.
BC77 BD BEA3    JSR      TESR      TIME OUT
BC7A 28 E7      BVC      SDBML2    LOOP IF NO TIME OUT
*
BC7C 31          INS      *SG*
BC7D 31          INS      *MG*
BC7E 31          INS      *HG*
BC7F 33          PUL B      ***
BC80 5A          DEC B
BC81 27 43      BEQ      SDBMX3    ONE LESS TIME OUT
BC83 37          PSH B      EXIT IF NO MORE
BC84 7C 0022    INC      DC1FLG    IND. READY TO TX
BC87 CE BF72    LDX      #ENDLIN   SEND END OF LINE
BC8A BD F7F5    JSR      TX2CHS    DO UNTIL
BC8D 20 CC      BRA      SDBML3    NO MORE TIME OUTS
*
BC8F CE 0691    SDBMS2 LDX      #BSOL+1 CHANGE LINE TO ASCII
BC92 BD F3EE    JSR      GTXB
BC95 3F          SWI
BC96 0000      FDB      0,ISTX2    TRANSMIT LINE
BC98 F404
BC9A 7F 0091    CLR      BSOLF     IND. BSOL NOT USED
*
BC9D 0F          SEI
BC9E BD F3F1    JSR      TMSR      GET CURRENT TIME
BCA1 0E          CLI
*
BCA2 96 88      SDBML4 LDA A  MLST      EXIT IF MAIN LINE
BCA4 81 FF      CMP A  #$FF      IS TO BE STOPPED
BCA6 27 C7      BEQ      SDBMXA
BCA8 86 05      LDA A  #$05      EXIT IF 5 MIN.
BCAA BD BEA3    JSR      TESR      HAVE EXPIRED
BCAD 29 C0      BVS      SDBMXA
BCAF 96 1E      LDA A  BUF2SR
BCB1 81 01      CMP A  #$01      WAIT UNTIL BUF2 EMPTY
BCB3 26 ED      BNE      SDBML4
BCB5 31          INS      *SG*
BCB6 31          INS      *MG*
BCB7 31          INS      *HG*
BCB8 31          INS      ***
BCB9 7E BC2D    JMP      SDBML      GET NEXT LINE
*
BCBC CE BF68    SDBMX2 LDX      #DF      DISPLAY "F"

```



```

BCBF 86 01          LDA A  #$01          FOR FRAMING ERROR
BCC1 C6 07          LDA B  #$07
BCC3 BD E7D6        JSR      DSCT
*
BCC6 BD FFF1 SDBMX3 JSR      DELAY2 SO MTS IS READY FOR
BCC9 02FF          FDB      $02FF NEXT LINE, WAIT 1.5SEC.
BCCB 96 86          LDA A  SYST IND. NOT TRANSMITTING
BCCD 84 F0          AND A  #$F0
BCCF 97 86          STA A  SYST
BCD1 BD E7DF        JSR      CFOF DISABLE TX2, COPY/FORT
BCD4 CE BD07        LDX      #TXSTBL SEND "* TRANSMISSION
BCD7 BD F7E3        JSR      TX1CHS STOPED '* BEL' "
BCDA 7F 0087        CLR      MLAC IND. MAIN NOT ACTIVE
BCDD 20 11          BRA      SYIDWL TO SYSTEM IDLE LOOP
*
BCDF BD FFF1 SDBMX1 JSR      DELAY2 SO TX OF PREVIOUS LINE
BCE2 01F2          FDB      $01F2 IS DONE, WAIT 1 SEC.
BCE4 BD E7DF        JSR      CFOF DISABLE TX2, COPY/FORT
BCE7 CE F7A5        LDX      #ENDTBL DISPLAY "END"
BCEA BD FFD6        JSR      DTSR2
BCED 7F 0087        CLR      MLAC IND. MAIN NOT ACTIVE
*
BCF0 01            SYIDWL NOP          SYSTEM IDLE WAIT LOOP
BCF1 01            NOP
BCF2 01            NOP
BCF3 01            NOP
BCF4 96 86          LDA A  SYST CHECK IF TRANSMISSION
BCF6 88 0F          EOR A  #$0F IS TO START
BCF8 85 0F          BIT A  #$0F
BCFA 01            NOP
BCFB 01            NOP
BCFC 26 F2          BNE      SYIDWL LOOP UNTIL IT IS
*
BCFE 7E BC1A        JMP      SDBM START TRANSMISSION
*
** END **
BD01 0006          RMB      6
*
BD07 TXSTBL EQU * TRANMISSION STOPED ( TABLE )
*
BD07 07 FCB $07 "BEL"
BD08 20 FCC ' * TRANMISSION STOPED *'
BD09 2A
BD0A 20
BD0B 54
BD0C 52
BD0D 41
BD0E 4E
BD0F 53
BD10 4D
BD11 49
BD12 53

```



```

BD13 53
BD14 49
BD15 4F
BD16 4E
BD17 20
BD18 53
BD19 54
BD1A 4F
BD1B 50
BD1C 45
BD1D 44
BD1E 20
BD1F 2A
BD20 07      FCB      $07,$0D,$0A,$7F,$04
BD21 0D                      "BEL","CR","LF","DEL",END
BD22 0A
BD23 7F
BD24 04
*
*
BD25 EPLF      EQU * ENSURE PROPER LINE FRAMING
*
* - THIS SUBROUTINE ENSURES THAT THE 24 BYTE LINE
*   IN BSOL STARTS WITH FRFR AND ENDS WITH FREN
* - IF THE LINE IS NOT FRAMED, DATA WILL BE READ
*   FROM THE BMS UNTIL A PROPER LINE IS PRODUCED
*   ( UPTO A MAXIMUM NUMBER OF BMS READS )
* - INPUTS: BSOL CONTENTS
*           DATA FROM BMS
* - OUTPUTS: Z IN CCR CLEAR IF THE LINE IN BSOL IS
*            PROPERLY FRAMED
*            V IN CCR SET IF BMS EMPTY (LINE NOT FRAMED)
* - CHANGES A,B,X,CCR, CONTENTS OF BSOL MAY CHANGE,
*   BMS STATUS MAY CHANGE
*
BD25 C6 04      LDA B      #$04      MAX. # REFRAMES
*
BD27 CE 0690 EPLFL4 LDX      #BSOL
BD2A A6 00      LDA A      0,X      IF FRONT END NOT
BD2C 81 80      CMP A      #FRFR    FRAMED, THEN GO FIND
BD2E 26 09      BNE        EPLFFF    FRONT END FRAME
BD30 A6 17      LDA A      23,X     IF BACK END NOT FRMD.
BD32 81 81      CMP A      #FREN    THEN FIND ANOTHER
BD34 26 03      BNE        EPLFFF    FRONT END FRAME
BD36 86 0F      LDA A      #$0F     Z=0,V=0(LINE FRAMED)
BD38 39          RTS
*
BD39 5A          EPLFFF DEC B      ANOTHER FR. ATTEMPT
BD3A 27 3A      BEQ        EPLFXE  EXIT IF TOO MANY REFR.
BD3C 86 80      LDA A      #FRFR    PREPARE TO FIND
BD3E CE 0690    LDX        #BSOL    THE FRAME FRONT
BD41 DF 8D      STX        EPLFX1
*
BD43 08          EPLFL1 INX          GET NEXT BYTE

```



```

BD44 8C 06A8      CPX      #BSOL+24  SKIP IF
BD47 27 17      BEQ      EPLFS1    NO FRFR IN LINE
BD49 A1 00      CMP      A 0,X      DO UNTIL
BD4B 26 F6      BNE      EPLFL1    FRAME FRONT FOUND
*
BD4D A6 00      EPLFL2 LDA      A 0,X      GET BYTE
BD4F 08          INX
BD50 DF 8F      STX      EPLFX2    SAVE NEXT BYTE LOC.
BD52 DE 8D      LDX      EPLFX1
BD54 A7 00      STA      A 0,X      MOVE BYTE UP
BD56 08          INX
BD57 DF 8D      STX      EPLFX1
BD59 DE 8F      LDX      EPLFX2    DO UNTIL FIRST
BD5B 8C 06A8    CPX      #BSOL+24  PART OF LINE
BD5E 26 ED      BNE      EPLFL2    MOVED UP
*
BD60 DF 8F      EPLFS1 STX      EPLFX2
BD62 37          EPLFL3 PSH      B      *** SAVE # RETRIES LEFT
BD63 D6 90      LDA      B EPLFX2+1  # BYTES MISSING
BD65 D0 8E      SUB      B EPLFX1+1  IN LINE
BD67 DE 8D      LDX      EPLFX1
BD69 BD B7E9    JSR      BMSO      GET REST OF LINE
BD6C 33          PUL      B      *** RESTORE # RETRIES
BD6D 28 03      BVC      EPLFS2    SKIP IF BMS NOT MT
BD6F 4F          CLR      A      Z=1(LINE NOT FRAMED)
BD70 0B          SEV          V=1 (BMS EMPTY)
BD71 39          RTS
*
BD72 27 EE      EPLFS2 BEQ      EPLFL3  RETRY, DATA NOT READ
*
BD74 20 B1          BRA      EPLFL4  DO UNTIL LINE FRAMED
*
BD76 4F          EPLFXE CLR      A      Z=1,V=0(LINE NOT FRAMED)
BD77 39          RTS
*
**  END  **
*
*
BD78 CFON      EQU *  COPY/FORTRAN ON
*
*  - A SUBROUTINE TO ACTIVATE OR START THE COPY
*  OF FORTRAN PROCESS IN MTS
*  - CHANGES A,X
*  - INPUTS : RXD - SPECIAL CHARACTERS RECIEVED
*              FROM MTS ( SEE "RX1" )
*              MLST - EXECUTION STOPS IF SET TO $FF
*              TPVEC - LOC. OF ASCII STRING TO
*                  REQUEST TRANSMISSION TO MTS
*  - OUTPUTS: ONLY TO MTS SYSTEM
*
BD78 07          TPA
BD79 36          PSH      A      *** SAVE INT. MASK
BD7A 0E          CLI          ALLOW INT. FOR TX
*

```



```

BD7B 7F 0049 CFONL2 CLR RXD
BD7E DE 92 LDX TPVEC GET TX. REQ. VECTOR
BD80 8C E75D CPX #COPYON CHECK VECTOR
BD83 27 08 BEQ CFONSO
BD85 8C E748 CPX #FORTON
BD88 27 03 BEQ CFONSO IF VECTOR IN ERROR,
BD8A CE E748 LDX #FORTON USE FORTRAN PROCESS
BD8D BD F7E3 CFONSO JSR TX1CHS SEND TRANSMIT REQ.
BD90 C6 50 LDA B #80 TIME OUT IN 8 SEC.
BD92 BD FFF1 CFONL1 JSR DELAY2
BD95 0030 FDB $0030 DELAY FOR .1 SEC.
BD97 96 88 LDA A MLST
BD99 81 FF CMP A #$FF EXIT IF
BD9B 27 25 BEQ CFONX MAINLINE TO STOP
BD9D 96 49 LDA A RXD
BD9F 81 3E CMP A #$3E SKIP IF ">" IS RXED
BDA1 27 05 BEQ CFONS1
BDA3 5A DEC B DO UNTIL
BDA4 26 EC BNE CFONL1 TIMEOUT OR ">"
BDA6 20 D3 BRA CFONL2 D.UN.MAIN STOP OR">"
*
BDA8 BD FFF1 CFONS1 JSR DELAY2 DELAY FOR .5 SEC.
BDAB 00F9 FDB $00F9
BDAD 7F 0049 CLR RXD
BDB0 CE F78C LDX #PTIPON SEND "%PTIP=ON"
BDB3 BD F7E3 JSR TX1CHS
BDB6 96 88 CFONL3 LDA A MLST
BDB8 81 FF CMP A #$FF EXIT IF
BDBA 27 06 BEQ CFONX MAIN TO BE STOPED
BDBC 96 49 LDA A RXD WAIT UNTIL
BDBE 81 11 CMP A #$11 "DC1" IS RXED
BDC0 26 F4 BNE CFONL3
*
BDC2 32 CFONX PUL A *** RESTORE INT. MASK
BDC3 06 TAP
BDC4 39 RTS
*
** END **
BDC5 0006 RMB 6
*
*
BDCB NOTX EQU * NO TRANSMISSION
*
* - A KEYBOARD FUNCTION TO STOP THE MAINLINE WHICH
* TRANSMITS DATA FROM THE BMS THE THE MTS SYSTEM
*
BDCB 96 87 LDA A MLAC EXIT IF MAINLINE
BDCD 27 04 BEQ NOTXX1 ALREADY INACTIVE
BDCF 86 FF LDA A #$FF IND. MAIN IS TO STOP
BDD1 97 88 STA A MLST
BDD3 3B NOTXX1 RTI
*
** END **
*
```



```

*
BDD4 SAVE EQU * SAVE
*
* - A KEYBOARD ACTIVATED FUNCTION TO SAVE THE
* STATUS OF DATA IN THE BMS IN NONVOLATILE STATUS
* PAGE OF THE BMS
* - THIS ROUTINE MUST BE EXECUTED BEFORE POWER DOWN
* IF THE STATE OF THE DATA IN THE BMS IS TO BE THE
* SAME AGAIN AFTER POWER UP
* - IF THIS ROUTINE IS NOT RUN BEFORE POWERDOWN,
* THEN:
* 1) IF NO DATA WAS TRANSFERED TO OR FROM THE
* BMS ( IF WE DIDN'T SAMPLE OR TRANSMIT )
* THEN IT MAKES NO DIFFERENCE
* 2) IF DATA WAS ONLY READ FROM THE BMS,( WE
* TRANSMIT ),THEN ON POWER UP, IT WOULD BE
* THE SAME AS IF NO DATA WAS READ
* ** 3) IF DATA WAS ONLY WRITTEN TO THE BMS,( WE
* SAMPLE ),THEN AFTER POWER UP IT WILL APPEAR
* THAT NO DATA WAS WRITTEN
* ** 4) IF DATA WAS WRITTEN AND READ FROM THE BMS,(WE
* SAMPLE AND TRANSMIT), THEN AFTER POWER UP,
* THE INTEGRITY OF ANY DATA THAT APPEARS TO BE
* IN THE BMS CANNOT BE GUARANTEED
*
BDD4 96 11 LDA A SECFLG CHECK IF 2ND FTN
BDD6 27 06 BEQ SAVES0 SELECTED
*
BDD8 BD FBF6 JSR SCNDSR
BDD8 7E B8EA JMP END GO TO 2ND FUNCTION
*
BDDE 96 86 SAVES0 LDA A SYST
BDE0 85 F0 BIT A #$F0
BDE2 27 07 BEQ SAVES1 SKIP IF NOT SAMPLING
*
BDE4 CE BF58 LDX #DUSEHS DISPLAY"USE .HS"TO
BDE7 BD FFD6 JSR DTSR2 PROMPT USER TO HALT
BDEA 3B RTI SAMPLING
*
BDEB 96 87 SAVES1 LDA A MLAC SKIP IF
BDED 27 07 BEQ SAVES2 MAIN NOT ACTIVE
*
BDEF CE BF60 LDX #DUSENT DISPLAY "USE NO TX"
BDF2 BD FFD6 JSR DTSR2 TO PROMPT USER TO
BDF5 3B RTI STOP TRANSMISSION
*
BDF6 96 89 SAVES2 LDA A FTNFLG SKIP IF A
BDF8 27 07 BEQ SAVES3 FUNCTION NOT ACTIVE
*
BDFA CE F423 LDX #ERRORT DISPLAY "ERROR" AS
BDFD BD FFD6 JSR DTSR2 A FTN ALREADY ACTIVE
BE00 3B RTI
*
BE01 BD FFD0 SAVES3 JSR CLDISP

```



```

BE04 CE BE30      LDX      #DSAVED      DISPLAY "SA"
BE07 86 02      LDA  A      #$02
BE09 5F          CLR  B
BE0A BD E7D6     JSR          DSCT
BE0D 7C 0089     INC      FTNFLG      IND. FUNCTION ACTIVE
BE10 0E          CLI          ALLOW FOR BMS INT.
BE11 BD E7DC     JSR          SPTP      SAVE PART.TXED PAGE
BE14 BD B7EF     JSR          BMWF      SAVE DATA IN BMIB
BE17 BD B7F2     JSR          BMSD      SAVE STATE OF BMS
BE1A CE BE30     LDX      #DSAVED      DISPLAY "SAVED"
BE1D BD FFD6     JSR          DTSR2
BE20 BD F7F2     JSR          CLBUF2     CLEAR TX BUFFER
BE23 7F 0091     CLR      BSOLF      IND. BSOL EMPTY
BE26 7F 0089     CLR      FTNFLG      IND. FTN NOT ACT.
BE29 3B          RTI

*
**      END      **
BE2A 0006          RMB      6

*
*
BE30 DSAVED EQU *      DISPLAY "SAVED"
*
BE30 6E          FCB      $6E          "S"
BE31 77          FCB      $77          "A"
BE32 33          FCB      $33          "V"
BE33 7A          FCB      $7A          "E"
BE34 1F          FCB      $1F          "D"
BE35 00          FCB      0,0,0        "  "
BE36 00
BE37 00
*
*
BE38 SPP1      EQU *      SAMPLE PROCESS ONE
*
*      - THE MAJOR ( AND DEFAULT ) SAMPLE PROCESS
*
BE38 0E          CLI          ALLOW INTERRUPTS
BE39 BD F3D6     JSR          SAMP      OBTAIN SAMPLE BLOCK
*
BE3C BD EFD0     JSR          FILT      FILTER SAMPLE,Y OUT
*
BE3F 7A 0084     DEC      SPLC      RESAMPLE DATA
BE42 27 28       BEQ      SPP1S1     SKIP,SAMPLE SELECTED
*
*      STORE PREVIOUSLY PREPARED DATA BLOCK
*
BE44 96 8C       LDA  A      BSILF      EXIT IF
BE46 2F 1F       BLE          SPP1X1     BSIL MT OR BMSI BUSY
BE48 86 F0       LDA  A      #$F0      IND. BSIL USED AND
BE4A 97 8C       STA  A      BSILF      BMSI BUSY
BE4C C6 18       LDA  B      #24        # OF BYTES IN BSIL
BE4E CE 0650     LDX      #BSIL
BE51 BD B7E6     JSR          BMSI      TRY STORING LINE
BE54 28 0C       BVC          SPP1S2     SKIP IF BMS NOT FULL

```



```

*
BE56 7F 008C      CLR      BSILF      IND.BSIL MT,BMSI IDLE
BE59 CE BF20      LDX      #DBMSFL  DISPLAY "BMS FULL"
BE5C BD FFD6      JSR      DTSR2
BE5F 7E B8DA      JMP      HSPL      HALT SAMPLING,EXIT
*
BE62 27 04      SPP1S2 BEQ      SPP1X2  EXIT,DATA NOT STORED
BE64 7F 008C      CLR      BSILF      IND.BSIL MT,BMSI IDLE
BE67 3B      SPP1X1 RTI
BE68 73 008C  SPP1X2 COM      BSILF      IND. BSIL USED AND
BE6B 3B      RTI      BMSI IDLE
*
BE6C SPP1S1 EQU *      PREPARE SAMPLE BLOCK
*
BE6C 96 85      LDA A  SPLCRS
BE6E 97 84      STA A  SPLC      RESET SAMPLE COUNTER
BE70 96 8C      LDA A  BSILF
BE72 27 07      BEQ      SPP1S3  SKIP, BSIL NOT FULL
*
BE74 CE E738      LDX      #DDATAL  DISPLAY"DATA.LST"AS
BE77 BD FFD6      JSR      DTSR2      THIS SAMPLE WAS LOST
BE7A 3B      RTI
*
BE7B CE 0490  SPP1S3 LDX      #Y  MOVE FILTER OUTPUT TO BSIL
BE7E BD F3F4      JSR      MDTI      ( 16 BYTES )
BE81 BD F3EB      JSR      CKSM  APP.CHECK SUM(2 BYTES)
BE84 BD F3E5      JSR      TMHB  ADD TIME ( 3 BYTES )
BE87 96 42      LDA A  FLGC
BE89 B7 0666      STA A  BSIL+22  ADD 1 FLAG BYTE
BE8C 7F 0042      CLR      FLGC
*
BE8F 86 80      LDA A  #FRFR      FRAME SAMPLE BLOCK
BE91 B7 0650      STA A  BSIL
BE94 86 81      LDA A  #FREN
BE96 B7 0667      STA A  BSIL+23
BE99 86 0F      LDA A  #$0F      IND. BSIL USED,
BE9B 97 8C      STA A  BSILF      BMSI IDLE
BE9D BD FFBE      JSR      FLSH      FLASH AN "S" IN
BEA0 6E      FCB      $6E,$06  LED #6
BEA1 06
BEA2 3B      RTI
*      ( 24 HEX BYTES TOTAL )
**      END      **
*
*
BEA3 TESR      EQU *      TIME EXPIRED SUBROUTINE
*
*      - A SUBROUTINE TO DETECT IF THE
*      CURRENT TIME IS BEOND AT LEAST A
*      CERTAIN NUMBER OF MINUTES PAST A
*      GIVEN TIME
*      - INPUTS :      HG  11,X      BCD VALUES
*                      MG  10,X      ( 3 BYTES
*                      SG   9,X      PUSHED ONTO THE

```



```

*          STACK BEFORE THIS ROUTINE IS
*          CALLED ) REPRESENTING A GIVEN TIME
*          ACC A          BCD VALUE OF AN
*          OFFSET OF 0 TO 10 MINUTES
* - OUTPUTS: V IN CCR SET ONLY IF THE
*          CURRENT TIME IS PAST THE GIVEN
*          TIME PLUS THE OFF SET
* - CHANGES A,B,X, V IN CCR
*
BEA3 34          DES          *HO* 6,X  SPACE ON STACK
BEA4 36          PSH A        *MO* 5,X  FOR OFFSET TIME
BEA5 34          DES          *SO* 4,X  (TIME OUT TIME)
BEA6 07          TPA
BEA7 36          PSH A        *** 3,X  AND TO SAVE CCR
BEA8 34          DES          *HC* 2,X
BEA9 34          DES          *MC* 1,X  AND CURRENT TIME
BEAA 34          DES          *SC* 0,X
BEAB 30          TSX          X PTS.TO ABOVE STACK
*
BEAC A6 0B          LDA A    11,X  FORM OFF SET TIME
BEAE A7 06          STA A    6,X  MOVE UP HOURS
BEB0 A6 09          LDA A    9,X  MOVE UP SECONDS
BEB2 A7 04          STA A    4,X
BEB4 A6 0A          LDA A    10,X  ADD MINUTES
BEB6 AB 05          ADD A    5,X
BEB8 19          DAA
BEB9 81 60          CMP A    #$60  CHECK FOR 60 MIN.
BEBB 0C          CLC
BEEC 2D 02          BLT      TESRS2 SKIP IF NO ROLL OVER
BEBE 8B A0          ADD A    #$A0  ROLL OVER(SETS CARRY)
BEC0 A7 05          TESRS2 STA A    5,X  SAVE MINUTES
BEC2 A6 06          LDA A    6,X  ADD CARRY TO HOURS
BEC4 89 00          ADC A    #$00
BEC6 19          DAA
BEC7 81 24          CMP A    #$24  ROLL OVER ON 24 HRS.
BEC9 26 01          BNE      TESRS3
BECB 4F          CLR A
BECC A7 06          TESRS3 STA A    6,X  SAVE HRS.
*
BECE 0F          SEI          DO NOT ALLOW INTS. TO
BECE BD F3F1      JSR          TMSR  GET CURRENT TIME
*
BED2 30          TSX          X PTS. TO STACK
BED3 A6 00          LDA A    0,X  SUB. OFF SET TIME FROM
BED5 A0 04          SUB A    4,X  CURRENT TIME, SECONDS
BED7 A6 01          LDA A    1,X  MINUTES
BED9 A2 05          SBC A    5,X
BEDB A6 02          LDA A    2,X  HOURS
BEDD A2 06          SBC A    6,X  A CARRY INDICATES
BEDF 25 10          BCS      TESRX1 TIME NOT EXPIRED
*
BEE1 A6 02          LDA A    2,X  IF CURRENT TIME NOT
BEE3 81 23          CMP A    #$23 NOT 23 HRS. WE HAVE
BEE5 26 04          BNE      TESRX2 OVER FLOW (OV.)

```



```

BEE7 A6 06          LDA A 6,X      IF OFF SET TIME 0 HRS.
BEE9 27 06          BEQ      TESRX1 WE DO NOT HAVE OV.
*
BEEB A6 03          TESRX2 LDA A 3,X      SET V IN RETURN
BEED 8A 02          ORA A  #$02      ( TIME EXPIRED )
BEEF 20 04          BRA      TESRX
*
BEF1 A6 03          TESRX1 LDA A 3,X      CLEAR V IN RETURN
BEF3 84 FD          AND A  #$FD      ( NO TIME OVERFLOW )
BEF5 06            TESRX  TAP          RESTORE CCR(INT.MASK)
BEF6 31            INS              *SC*
BEF7 31            INS              *MC*
BEF8 31            INS              *HC*
BEF9 31            INS              ***
BEFA 31            INS              *SO*
BEFB 31            INS              *MO*
BEFC 31            INS              *HO*
BEFD 39            RTS
*
**  END  **
*
*  TABLES
*
BF20          ORG      $BF20
*
BF20 DBMSFL EQU *  DISPLAY BUBBLE MEM. SYST. FULL
*
BF20 BE          FCB      $BE          "B."
BF21 16          FCB      $16          "M."
BF22 86          FCB      $86          "
BF23 EE          FCB      $EE          "S."
BF24 72          FCB      $72          "F"
BF25 1C          FCB      $1C          "U"
BF26 38          FCB      $38          "L"
BF27 38          FCB      $38          "L"
*
*
BF28 DUSEL EQU *  DISPLAY USE SELECT
*
BF28 3D          FCB      $3D          "U"
BF29 6E          FCB      $6E          "S"
BF2A 7A          FCB      $7A          "E"
BF2B 00          FCB      $00
BF2C 6E          FCB      $6E          "S"
BF2D 7A          FCB      $7A          "E"
BF2E 38          FCB      $38          "L"
BF2F 00          FCB      $00
*
*
BF30 DTTXSA EQU *  DISPLAY TRY TX OR SAVE
*
BF30 45          FCB      $45          "T"
BF31 12          FCB      $12          "R"

```


BF32	2F	FCB	\$2F	"Y"
BF33	00	FCB	\$00	" "
BF34	45	FCB	\$45	"T"
BF35	B5	FCB	\$B5	"X."
BF36	6E	FCB	\$6E	"S"
BF37	F7	FCB	\$F7	"A."

*

*

BF38 DSNDG EQU * DISPLAY SENDING

*

BF38	6E	FCB	\$6E	"S"
BF39	7A	FCB	\$7A	"E"
BF3A	16	FCB	\$16	"N"
BF3B	1F	FCB	\$1F	"D"
BF3C	04	FCB	\$04	"I"
BF3D	16	FCB	\$16	"N"
BF3E	7C	FCB	\$7C	"G"
BF3F	00	FCB	\$00	" "

*

*

BF40 DNOTRS EQU * DISPLAY NO TRANSMISSION

*

BF40	16	FCB	\$16	"N"
BF41	9E	FCB	\$9E	"O."
BF42	45	FCB	\$45	"T"
BF43	12	FCB	\$12	"R"
BF44	77	FCB	\$77	"A"
BF45	16	FCB	\$16	"N"
BF46	EE	FCB	\$EE	"S."
BF47	00	FCB	\$00	" "

*

*

BF48 DBSMT EQU * DISPLAY BUBBLE SYSTEM EMPTY

*

BF48	BE	FCB	\$BE	"B."
BF49	EE	FCB	\$EE	"S."
BF4A	00	FCB	\$00	" "
BF4B	16	FCB	\$16	"M"
BF4C	06	FCB	\$06	" "
BF4D	45	FCB	\$45	"T"
BF4E	00	FCB	\$00	" "
BF4F	00	FCB	\$00	" "

*

*

BF50 DTRLTR EQU * DISPLAY TRY LATER

*

BF50	45	FCB	\$45	"T"
BF51	12	FCB	\$12	"R"
BF52	AF	FCB	\$AF	"Y."
BF53	38	FCB	\$38	"L"
BF54	77	FCB	\$77	"A"
BF55	45	FCB	\$45	"T"
BF56	7A	FCB	\$7A	"E"
BF57	12	FCB	\$12	"R"


```

*
BF58 DUSEHS EQU *   DISPLAY USE HALT SAMPLING
*
BF58 3D             FCB      $3D      "U"
BF59 6E             FCB      $6E      "S"
BF5A 7A             FCB      $7A      "E"
BF5B 80             FCB      $80      " "
BF5C 37             FCB      $37      "H"
BF5D 6E             FCB      $6E      "S"
BF5E 00             FCB      $00      " "
BF5F 00             FCB      $00      " "
*
*
BF60 DUSENT EQU *   DISPLAY USE NO TRANSMISSION
*
BF60 3D             FCB      $3D      "U"
BF61 6E             FCB      $6E      "S"
BF62 7A             FCB      $7A      "E"
BF63 80             FCB      $80      " "
BF64 16             FCB      $16      ".N"
BF65 9E             FCB      $9E      "O."
BF66 45             FCB      $45      "T"
BF67 35             FCB      $35      "X"
*
*
          BF68      DF      EQU      *           DISPLAY "F"
*
BF68 72             FCB      $72      "F"
*
*
BF69 ECHOFF EQU *   ECHO OFF ( FOR HALF DUPLEX )
BF69 25             FCC      '%ECHO=OFF'
BF6A 45
BF6B 43
BF6C 48
BF6D 4F
BF6E 3D
BF6F 4F
BF70 46
BF71 46
BF72 0D             ENDLIN FCB      $0D,$0A,$7F,$04
BF73 0A
BF74 7F
BF75 04
*
*
BF76 SIGID EQU *   SIGNON ID
BF76 53             FCC      'SIG '
BF77 49
BF78 47
BF79 20
BF7A 0004           RMB 4   LEAVE SPACE FOR ID IN EPROM
BF7E 0D             FCB      $0D,$0A,$7F,$04
BF7F 0A

```



```

BF80 7F
BF81 04
*
*
BF82 PASSWD EQU * PASSWORD
*
BF82 0006 RMB 6 LEAVE SPACE FOR PASSWORD IN EPROM
BF88 0D FCB $0D,$0A,$7F,$04
BF89 0A
BF8A 7F
BF8B 04
*
* JUMP TABLE FOR ROM 8
*
BFD0 ORG $BFD0
*
BFD0 7E BE38 SPP1V JMP SPP1
BFD3 7E B81E SPLP2V JMP SPLP2
BFD6 7E BAAD JMP SEL
BFD9 7E BDD4 JMP SAVE
BFDC 7E BBD3 JMP TX
BFDF 7E B916 JMP SIG
BFE2 7E B854 JMP SSPL
BFE5 7E BCF0 JMP SYIDWL
END

```


Appendix C -- Fortran Support

SPRNTD

C FILE SPRNTD

C

C A PROGRAM TO PRINT NUMERICAL DATA IN SECTIONS
C OF A DATA FILE GENERATED BY THE DATA AQUISITION
C SYSTEM

C

C TO COMPILE TYPE:

C R *FORTG SCARDS=SPRNTD SPUNCH=PRNTD T=10
C (REQUITED ONLY ONCE IF PRNTD IS SAVED)

C

C TO RUN TYPE:

C R PRNTD 1=(DATA FILE NAME) [6=OUTPUT DEVICE NAME] T=30
C [OPTIONAL]

C

CALL PRNTD

STOP

END

C

C

C

SUBROUTINE PRNTD

C

C A SUBROUTINE TO PRINT OUT NUMERICAL DATA
C FROM ANY PART OF A FILE GENERATED BY THE MICRO
C BASED DATA ACQUISITION SYSTEM.

C

C NDATA - NUMERICAL DATA (SAMPLE)

C CDATA - CHARACTER DATA

C TDATA - DATA TYPE, ONE OF THE ABOVE

C FLAG - SOME INFORMATION ABOUT THE DATA LINE

C TIME- A VECTOR INDICATING WHEN THE SAMPLE

C WAS TAKEN. THE FIRST ELEMENT IS HOURS.

C THE NEXT IS MINUTES,THE LAST IS SECONDS

C HVCL1,2 - HEXADECIMAL INPUT VALUES OF CALIBRATION
C LEVELS

C AVCL1,2 - ADJUSTED (PROPER NUMERIC) VALUES OF
C CALIBRATION LEVELS

C CHTP - INDICATES THE TYPE OF EACH CHANNEL

C CAL - INDICATES THE CALIBRATION STATUS OF EACH
C CHANNEL

C ISTART,IEND,ILINEN - FIRST, LAST, AND NEXT LINES
C TO BE READ IN FILE

C

REAL NDATA(8),AVCL1(8),AVCL2(8),D

INTEGER*2 TDATA,HVCL1(8),HVCL2(8),FLAG(2),CHTP(8),

* CAL(8),I,J,TIME(3)

INTEGER ILINEN,ISTART,IEND


```

LOGICAL*1 CDATA(80),STRING(6),CHAR(4,9)
*      /'M',' ','M',' ','I','C','I',' ','L','M',
*      'L','V','I',' ','I','O','V','H','M','L',
*      'O','2','E','T','L','O','T','S','T',' ',
*      ' ',' ','S',' ',' ',' ',' /
C
C      DETERMINE WHAT DATA IS TO BE READ
C
      WRITE(5,9)
9  FORMAT('OENTER FILE NAME.  (6A1)')
      READ(5,8) STRING
8  FORMAT(6A1)
C
1000 REWIND 1
C
      ISTART=1
      ILINEN=1
      IEND=1000000
      WRITE(5,10)
10  FORMAT('OPRESS "RETURN" IF YOU WISH TO PRINT THE' /
*      ' COMPLETE FILE.' /
*      ' ENTER      1 IF YOU WISH TO PRINT ONLY' /
*      ' PART OF THE FILE.  (I1)')
      READ(5,11) I
11  FORMAT(I1)
      IF(I.EQ.0) GO TO 1100
1010 WRITE(5,12)
12  FORMAT('OENTER FIRST LINE TO BE PRINTED' /
*      ' WITH A DECIMAL POINT BEHIND IT. (F9.0)')
      READ(5,13) D
13  FORMAT(F9.0)
      IF(D)1010,1010,1020
1020 ISTART=D
      WRITE(5,7) ISTART
7  FORMAT(' ',I10)
1030 WRITE(5,14)
14  FORMAT('OENTER LAST LINE TO BE PRINTED' /
*      ' WITH A DECIMAL POINT BEHIND IT. (F9.0)')
      READ(5,13) D
      IF(D)1030,1030,1040
1040 IEND=D
      WRITE(5,7) IEND
C
C      GET CHANNEL TYPES AND CALIBRATION POINTS
C
1100 DO 1190 I=1,8
1190 CALL PRERD(HVCL1,HVCL2,AVCL1,AVCL2,I,CHTP,CAL)
C
C      PRINT DATA FILE SPECS.
C
      WRITE(6,17) STRING
17  FORMAT('1 PRINTOUT FROM DATA FILE ',6A1)
      WRITE(6,18) ISTART,IEND
18  FORMAT('O FROM LINE ',I7/

```



```

      *          '      TO LINE ',I7)
      WRITE(6,15)
15  FORMAT(' 0' ,5X,' #1' ,10X,' #2' ,10X,' #3' ,10X,' #4' ,10X,
      *          ' #5' ,10X,' #6' ,10X,' #7' ,10X,' #8' ,6X,
      *          ' FLAG' ,4X,' TIME      LINE NUMBER' )
      WRITE(6,16) ((CHAR(CHTP(I)),J),J=1,9),I=1,8)
16  FORMAT(' ',8(1X,9A1,2X),2X,2Z1///)

C
C      NOW PRINT DATA
C
C
C      ASSUME NUMERIC DATA
      TDATA=1
1200 CALL RDCAL(HVCL1,HVCL2,AVCL1,AVCL2,CAL,ISTART,
      *          IEND,ILINEN,NDATA,CDATA,TDATA,FLAG,TIME)
      IF(TDATA.LT.1) GO TO 1300
      GO TO (1220,1250,1220),TDATA
      GO TO 1300
1220 WRITE(6,19) NDATA,FLAG,TIME,ILINEN
      19  FORMAT(' ',8(F10.4,2X),2X,2Z1,4X,2(I2,' :'),I2,I12)
      GO TO 1200

C
C      ASSUME CHARACTER DATA
1250 WRITE(6,20) CDATA
      20  FORMAT(' ',80A1)
1260 CALL RDCAL(HVCL1,HVCL2,AVCL1,AVCL2,CAL,ISTART,
      *          IEND,ILINEN,NDATA,CDATA,TDATA,FLAG,TIME)
      WRITE(6,20) CDATA
      IF(TDATA) 1300,1300,1270
1270 GO TO (1200,1260,1200),TDATA
      GO TO 1300

C
C      IS ANYTING ELSE IN FILE TO BE PRINTED?
C
C
1300 WRITE(5,21)
      21  FORMAT(' OPRESS "RETURN" IF NO OTHER DATA IN' /
      *          ' FILE IS TO BE PRINTED.' /
      *          ' ENTER      1      TO PRINT OTHER DATA' /
      *          ' IN THIS FILE.' )
      READ(5,11) I
      IF(I)1350,1350,1000
1350 RETURN
      END

C
C
C      SUBROUTINE PRERD(HVCL1,HVCL2,AVCL1,AVCL2,CHNO,
      *          CHTP,CAL)

C
C      THIS ROUTINE PREPARES TO READ DATA FROM A
C      PARTICULAR CHANNEL IN A DATA FILE GENERATED BY THE
C      MICRO DATA ACQUISITION SYSTEM BY SPECIFYING
C      THE CHANNEL TYPE AND CALIBRATION POINTS
C

```



```

C      HVCL1,2 - HEXADECIMAL INPUT VALUES OF CALIBRATION
C              LEVELS
C      AVCL1,C - ADJUSTED(PROPER NUMERICAL)VALUES OF
C              THOSE CALIBRATION LEVELS
C      CHNO - THE NUMBER OF THE CHANNEL TO BE PREPARED
C      CHTP - INDICATES THE TYPE OF EACH CHANNEL
C      CAL - INDICATES THE CALIBRATION STATUS OF EACH
C           CHANNEL
C
C      INTEGER*2 HVCL1(8),HVCL2(8),CHNO,CHTP(8),CAL(8),I
C      REAL AVCL1(8),AVCL2(8)
C      LOGICAL*1 MV(13) / 'I', 'N', ' ', 'M', 'I', 'L', 'I', 'V', 'O',
*                      'L', 'T', 'S', ' ', ' ', ' ', ' ', ' ', ' ',
*      CMH2O(13) / 'I', 'N', ' ', ' ', ' ', ' ', 'C', 'M', ' ', ' ', ' ', 'H',
*                      '2', 'O', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
*      MM(13) / 'I', 'N', ' ', ' ', 'M', 'I', 'L', 'I', 'M', 'E',
*                      'T', 'E', 'R', 'S', ' ', ' ', ' ', ' ', ' ', ' ',
*      BB(13) / ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
*                      ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
C
C      GET TYPE OF CHANNEL
C
C      1010 WRITE(5,12) CHNO
C      12 FORMAT('0FOR CHANNEL #',I1/
C      *      ' ENTER THE CANNEL TYPE. (I1)' /
C      *      ' 1 FOR ELECTRICAL DATA' /
C      *      ' 2 FOR PRESSURE DATA' /
C      *      ' 3 FOR DISPLACEMENT DATA' /
C      *      ' 4 FOR GENERAL DATA' )
C      READ(5,11)CHTP(CHNO)
C      11 FORMAT(I1)
C      IF(CHTP(CHNO)) 1010,1010,1015
C      1015 I=CHTP(CHNO)
C      GO TO (1100,1200,1300,1400),I
C      GO TO 1010
C
C      ASSIGN CALIBRATION VALUES
C
C      ELECTRICAL CALIBRATION DATA (IN MILIVOLTS)
C      1100 HVCL1(CHNO)=0
C      HVCL2(CHNO)=- (8188*.1/2.5 + .5)
C      AVCL1(CHNO)=0.0
C      AVCL2(CHNO)=-100
C      CALL CDFLT(HVCL1,HVCL2,AVCL1,AVCL2,MV,CHNO)
C      GO TO 1800
C
C      PRESSURE CALIBRATION DATA (IN CM H2O)
C      1200 HVCL1(CHNO)=0
C      HVCL2(CHNO)=8188*.1/2.5 + .5
C      AVCL1(CHNO)=0.0
C      AVCL2(CHNO)=100
C      CALL CDFLT(HVCL1,HVCL2,AVCL1,AVCL2,CMH2O,CHNO)
C      GO TO 1800

```



```

C
C      DISPLACEMENT DATA  (IN MILIMETERS)
1300 HVCL1(CHNO)=0
      HVCL2(CHNO)=8188*.1/2.5  + .5
      AVCL1(CHNO)=0.0
      AVCL2(CHNO)=5
      CALL CDFLT(HVCL1,HVCL2,AVCL1,AVCL2,MM,CHNO)
      GO TO 1800
C
C GENERAL CALIBRATION(OUTPUT SAME VOLTAGE AS ANALOG INPUT)
1400 HVCL1(CHNO)=0
      HVCL2(CHNO)=8188/2.5  + .5
      AVCL1(CHNO)=0.0
      AVCL2(CHNO)= 1.0
      CALL CDFLT(HVCL1,HVCL2,AVCL1,AVCL2,BB,CHNO)
C
C      INDICATE THIS CHANNEL INITIALLY CALIBRATED
1800 CAL(CHNO)=1
      RETURN
      END
C
C
      SUBROUTINE CDFLT(HVCL1,HVCL2,AVCL1,AVCL2,CF,CN)
C
C      ASKS IF DEFALT CALIBRATION POINTS ARE TO BE CHANGED
C
C      INPUTS ONLY:
C      CF - CHARACTER FIELD EXPLAINING UNITS
C      CN - CHANNEL NUMBER IN QUESTION
C      INTEGER*2 CN
C      LOGICAL*1 CF(13)
C
C      INPUTS AND OUTPUTS
C      HVCL1,2 - HEXADICIMAL VALUES OF CALIBRATION LEVELS
C      AVCL1,2 - ADJUSTED(PROPER NUMERICAL)VALUES OF SAME
C      INTEGER*2 HVCL1(8),HVCL2(8)
C      REAL AVCL1(8),AVCL2(8)
C
C      INTERNAL ONLY
C      INTEGER INST
C
100 WRITE(5,10) CN,HVCL1(CN),AVCL1(CN),CF,
*           HVCL2(CN),AVCL2(CN),CF
10 FORMAT('0 THE CALIBRATION POINTS FOR CHANNEL #',I1/
*        ' ARE :'/
*        ' ',5X,Z4,' IN HEX.,   FOR ',F10.4,2X,13A1/
*        ' ',5X,Z4,' IN HEX.,   FOR ',F10.4,2X,13A1/
*        '0ENTER:  (I1)'/
*        ' "RETURN" FOR NO CHANGE' /
*        '      1   TO CHANGE CALIBRATION LEVELS' /
*        '      2   TO CHANGE THE HEX. INPUT VALUES' )
      READ(5,11) INST
11 FORMAT(I1)
      IF(INST)100,200,110

```



```

110 GO TO (120,130),INST
    GO TO 100

```

```

C
120 WRITE(5,12) CF
12 FORMAT('0ENTER FIRST CALIBRATION LEVEL ',13A1/
*      ' (F10.4)')
    READ(5,13) AVCL1(CN)
13 FORMAT(F10.4)
    WRITE(5,14) CF
14 FORMAT('0ENTER SECOND CALIBRATION LEVEL ',13A1/
*      ' (F10.4)')
    READ(5,13) AVCL2(CN)
    GO TO 100

```

```

C
130 WRITE(5,15)
15 FORMAT('0ENTER HEXADECIMAL INPUT VALUE FOR' /
*      ' FIRST CALIBRATION LEVEL, (Z4)')
    READ(5,16) HVCL1(CN)
16 FORMAT(Z4)
    WRITE(5,17)
17 FORMAT('0ENTER HEXADECIMAL INPUT VALUE FOR' /
*      ' SECOND CALIBRATION LEVEL, (Z4)')
    READ(5,16) HVCL2(CN)
    WRITE(5,18) CN,CN
18 FORMAT('0  IF A "C',I1,' " OR A "D',I1,' " FLAG' /
*      ' IS ENCOUNTERED IN THE DATA FILE,THEN THE FIRST' /
*      ' OR SECOND HEX. INPUT VALUE WILL BE CHANGED.' /
*      ' THESE FLAGS ARE ENTERED INTO THE DATA FILE' /
*      ' BY USEING THE CALIBRATE FUNCTION, ("CAL"),' /
*      ' ON THE MICRO BASED DATA ACQUISITION SYSTEM.')
    GO TO 100
200 RETURN
    END

```

```

C
C
    SUBROUTINE INCAL(HVCL1,HVCL2,AVCL1,AVCL2,CAL)
C
C    THIS SUBROUTINE INITIALIZES THE CALIBRATION
C    POINTS OF ALL CHANNELS SO "RDCAL" CAN RUN
C    EVEN IF THE CHANNELS MAY NOT BE CALIBRATED
C
C    INTERNAL ONLY:
    INTEGER I
C
C    OUTPUTS ONLY:
    HVCL1,2 - HEX. VALUES OF TWO CALIBRATION POINTS
              FOR EACH CHANNEL
    AVCL1,2 - ADJUSTED(PROPER NUMERICAL)VALUE OF SAME
    CAL - INDICATES THE STATUS OF CALIBRATION
           OF EACH CHANNEL (SET TO INITIALIZED ONLY)
C
    INTEGER*2 HVCL1(8),HVCL2(8),CAL(8)
    REAL AVCL1(8),AVCL2(8)
    DO 100 I=1,8

```



```

      HVCL1(I)=0
      HVCL2(I)=8188/2.5 + .5
      AVCL1(I)=0.0
      AVCL2(I)=1.0
100 CAL(I)=0
      RETURN
      END

C
C
      SUBROUTINE RDCAL(HVCL1,HVCL2,AVCL1,AVCL2,CAL,ISTART,
*                   IEND,ILINEN,NDATA,CDATA,TDATA,FLAG,TIME)

C
C      THIS ROUTINE READ DATA FROM A FILE GENERATED BY THE
C      MICRO BASED DATA ACQUISITION SYSTEM. NUMERICAL DATA
C      IS CALIBRATED FOR EACH CHANNEL BY THE USE OF TWO
C      CALIBRATION POINTS. CHARACTER DATA IS UNCHANGED
C
C      INPUTS ONLY:
C      AVCL1,2 - ACTUAL(PROPER NUMERICAL)VALUES OF THE
C                CALIBRATION POINTS
C      ISTART,IEND - START AND END LINE NUMBERS IN FILE
C                   OF DATA TO BE READ
C      REAL AVCL1(8),AVCL2(8)
C      INTEGER ISTART,IEND

C
C      INPUTS AND OUTPUTS:
C      HVCL1,2 - HEXADECIMAL INPUT VALUES OF THE CALIBRATION
C                POINTS
C      CAL - INDICATES THE CALIBRATION STATUS OF EACH
C            CHANNEL
C      ILINEN - THE NEXT LINE TO BE READ IN FILE
C      TDATA = 0 IF NO MORE DATA IN FILE (NO OUTPUT)
C            = 1 IF NEXT LINE IS NUMERIC (SAMPLE) DATA
C            = 2 IF NEXT AND CURRENT LINES ARE CHARACTER DATA
C            = 3 IF NEXT LINE IS NUMERIC DATA AND THE
C                CURRENT LINE IS CALIBRATION DATA
C      IF ON ENTRY WE ARE EXPECTING A CHARACTER LINE
C      WE CAN ONLY OUTPUT A CHARACTER LINE.
C      IF ON ENTRY WE ARE EXPECTING A NUMERIC LINE,
C      THEN ON OUTPUT, IF TDATA=1,3 WE ARE OUTPUTTING
C      A NUMERIC LINE, OR IF TDATA=2 WE ARE OUTPUTTING
C      A CHARACTER LINE.
C
C      ANY CHARACTER DATA ENTERED INTO THE FILE SHOULD
C      BE PRECEDED WITH A BLANK LINE THAT ONLY
C      CONTAINS A HEXADECIMAL DIGIT OTHER THAN 0
C      IN THE FIRST COLUMN. ANY LINES OF CHARACTER DATA
C      SHOULD HAVE A HEXADICIMAL CHARACTER OTHER THAN
C      0 IN THE FIRST COLUMN. THE LAST LINE OF A
C      NUMBER OF CHARACTER LINES MUST CONTAIN A 0
C      OR A BLANK IN THE FIRST COLUMN.
C
C      1
C      1 THIS IS AN EXAMPLE OF A CHARACTER DATA
C      1 BLOCK THAT CAN OCCUR ANYWHERE IN THE DATA

```



```

C      1  FILE. A NORMAL LINE CONTAINS A HEXADECIMAL
C      1  CHARACTER OTHER THAN ZERO IN THE FIRST
C      1  COLUMN WHICH IS NOT PASSED AS DATA FROM
C      1  THIS ROUTINE. THE LAST LINE MUST CONTAIN
C      0  A ZERO OR A BLANK IN THE FIRST COLUMN.
C
C      * FIRST COLUMN IN FILE
C
C      INTEGER*2 HVCL1(8),HVCL2(8),CAL(8),TDATA
C      INTEGER ILINEN
C
C      OUTPUTS ONLY:
C      NDATA - CALIBRATED NUMERIC DATA
C      FLAG - SOME INFORMATION ABOUT THE DATA
C      TIME - VECTOR OF THE HOUR,MINUTE,AND SECOND
C              THE SAMPLE WAS TAKEN
C      CDATA - CHARACTER DATA
C
C      REAL NDATA(8)
C      INTEGER*2 FLAG(2),TIME(3)
C      LOGICAL*1 CDATA(80),BLANK/' '/
C
C      INTERNAL ONLY:
C      S - SAMPLE DATA VECTOR (HEX.)
C      CC - INDICATES IF A LINE IS CHARACTER DATA
C      INTEGER*2 S(8),CC,I
C      REAL B,D,M
C      LOGICAL*1 CHAR(80)
C
C      DO 800 I=1,8
800  NDATA(I)=0.0
C
C      DO 900 I=1,80
900  CDATA(I)=BLANK
C
C      1000 IF(ILINEN.GE.ISTART) GO TO 1200
C
C      SCAN PREVIOUS LINES FOR CALIBRATION DATA
C      IF(TDATA.EQ.2) GO TO 1100
C
C      READ NUMERIC DATA
C      READ(1,10,END=1900) CC,S,TIME,FLAG
10  FORMAT(Z1,8Z4,4X,3I2,2Z1)
C      ILINEN=ILINEN+1
C
C      IF(CC.NE.0) GO TO 1090
C      TDATA=1
C      IF(FLAG(1).NE.12) GO TO 1050
C      IF(FLAG(2).LT.1 .OR. FLAG(2).GT.8) GO TO 1000
C      HVCL1(FLAG(2))=S(FLAG(2))
C      IF(CAL(FLAG(2)).EQ.0) CAL(FLAG(2))=1
C      IF(CAL(FLAG(2)).EQ.2) CAL(FLAG(2))=3
C      GO TO 1000
C
C

```



```

1050 IF(FLAG(1).NE.13) GO TO 1000
      IF(FLAG(2).LT.1 .OR. FLAG(2).GT.8) GO TO 1000
      HVCL2(FLAG(2))=S(FLAG(2))
      IF(CAL(FLAG(2)).EQ.0) CAL(FLAG(2))=2
      IF(CAL(FLAG(2)).EQ.1) CAL(FLAG(2))=3
      GO TO 1000
C
C      INDICATE CHARACTER DATA
1090 TDATA=2
      GO TO 1000
C
C      READ CHARACTER DATA
1100 READ(1,11,END=1900) CC,CHAR
      11 FORMAT(Z1,80A1)
      ILINEN=ILINEN+1
C
      IF(CC.NE.0) GO TO 1090
      TDATA=1
      GO TO 1000
C
C      OUTPUT A LINE OF DATA
C
1200 IF(ILINEN.GT.IEND) GO TO 1900
C
      IF(TDATA.NE.2) GO TO 1250
C
C      WE ARE EXPECTING CHARACTER DATA
      READ(1,11,END=1900) CC,CDATA
      ILINEN=ILINEN+1
      TDATA=2
      IF(CC.EQ.0) TDATA=1
      RETURN
C
C      WE ARE EXPECTING NUMERIC DATA
1250 READ(1,10,END=1900) CC,S,TIME,FLAG
      ILINEN=ILINEN+1
      IF(CC.EQ.0) GO TO 1260
      TDATA=2
      RETURN
C
C      CHECK FOR CALIBRATION DATA
1260 TDATA=1
      IF(FLAG(1).NE.12) GO TO 1270
      IF(FLAG(2).LT.1 .OR. FLAG(2).GT.8) GO TO 1280
      HVCL1(FLAG(2))=S(FLAG(2))
      TDATA=3
      GO TO 1280
C
1270 IF(FLAG(1).NE.13) GO TO 1280
      IF(FLAG(2).LT.1 .OR. FLAG(2).GT.8) GO TO 1280
      HVCL2(FLAG(2))=S(FLAG(2))
      TDATA=3
C
C      CALIBRATE ALL CHANNELS

```



```
1280 DO 1290 I=1,8  
      D=HVCL2(I)-HVCL1(I)  
      B=(HVCL2(I)*AVCL1(I)-HVCL1(I)*AVCL2(I))/D  
      M=(AVCL2(I)-AVCL1(I))/D  
1290 NDATA(I)=M*S(I)+B  
      RETURN
```

```
C  
C      EXIT FOR NO MORE DATA FROM FILE  
C
```

```
1900 TDATA=0  
      RETURN  
      END
```


SPROCESS

```

C      FORTG      FILE: SPROCESS
C      CALLING SEQUENCE: R PROCESS 6=(STORAGE FILE) T=30
C      INPUTS      FROM LOGICAL UNIT 5=*SOURCE*
                  INTEGER*2  S1,S2,S3,S4,S5,S6,S7,S8,CHAR(4),CS
                  LOGICAL*1  H(2),M(2),S(2),FFTF/.FALSE./
                  LOGICAL*1  CMNT/.FALSE./
                  LOGICAL*1  STRING(80)
C      OTHER VARIABLES
                  INTEGER*2  F/'F'/,CCS
                  REAL  FFTM(8,256)
C      OUTPUTS
C      OUTPUT TO RAW DATA FILE ON LU. 6
C      FFTM, AND DATA TO L.U. 7
1000  WRITE(5,15)
15    FORMAT(' >>' )
100   GO TO 500
400   WRITE(5,14)
14    FORMAT(' RR' )
500   READ(5,10,END=200) CMNT,S1,S2,S3,S4,S5,S6,S7,S8,
*                                     CS,H,M,S,CHAR
10    FORMAT(Z1,9Z4,6A1,4A1)
      CMNT=.NOT.CMNT
      IF( CMNT ) GO TO 150
      GO TO 300
150   WRITE(6,11) S1,S2,S3,S4,S5,S6,S7,S8,CS,H,M,S,CHAR
11    FORMAT(' ',8Z4,Z4,6A1,4A1)
C      ADD THE 16 BYTES REP. BY S1,...,S8 TO CALULATE THE
C      CHECK SUM VALUE, CCS.
      CCS=S1+S2+S3+S4+S5+S6+S7+S8
      IF ( CHAR(4) .EQ. F ) FFTF=.TRUE.
      IF (CS.EQ.CCS) GO TO 600
      WRITE(6,13) CS,CCS
13    FORMAT('          ERROR          CS TXED=',Z4,' CS COMP=',Z4)
      GO TO 500
C600  WRITE(7,16) S1,S2,S3,S4,S5,S6,S7,S8,H,M,S,CHAR
16    FORMAT(' ',8I6,1X,2(2A1,' :'),2A1,1X,4A1)
600   GO TO 100
300   WRITE(6,17)
17    FORMAT(' 1' )
310   READ(5,18,END=200) CMNT,STRING
18    FORMAT(Z1,80A1)
      WRITE(6,18) CMNT,STRING
      IF( CMNT ) GO TO 310
      GO TO 100
200   STOP
      END

```


SDFIL

```

C   FORTRAN PROGRAM TO CALCULATE DIGITAL FILTER
C   COEFFICIENTS FOR A 4TH ORDER LOW PASS BUTTERWORTH
C
C   RESULTING FILTER :   H(Z) = H1(Z)H2(Z)
C
C           A0*Z*Z + A1*Z + A2
C   H1(Z)  =  -----
C           Z*Z    + B1*Z + B2
C
C           C0*Z*Z + C1*Z + C2
C   H2(Z)  =  -----
C           Z*Z    + D1*Z + D2
C
C
C   REAL*8 AAA,BBB,ALPHA,THETAP,THETAC,ALPHA2,THETAF
C   REAL*8 AA0,AA1,AA2,BB0,BB1,BB2
C   REAL*8 A0,A1,A2,B0,B1,B2,C0,C1,C2,D0,D1,D2,RB1,RB2,
C   *      RD1,RD2
C   REAL*8 T,T2,RESP*4(300)
C   INTEGER*2 IA0,IA1,IA2,IB1,IB2,IC0,IC1,IC2,ID1,ID2
C   COMPLEX*16 R1,R2
C   INTEGER  W,O,P,Q,R,Y,SS
C   INTEGER*2 S,N(300),IRESP(300)
C   AAA=.76536686D 00
C   BBB=.184775907D 01
C   WRITE(6,10)
10  FORMAT('0 ENTER SAMPLE PERIOD, T , IN SECONDS' /
C   *      '(F10.6)')
C   READ(5,11) T
11  FORMAT(F10.6)
C   WRITE(6,11) T
C   WRITE(6,12)
12  FORMAT('0 ENTER CUT OFF FREQUENCY IN HERTZ' /
C   *      '(F9.4)')
C   READ(5,13) THETAF
13  FORMAT(F9.4)
C   WRITE(6,11) THETAF
C   THETAC=THETAF*2*3.141592654D 00*T
C   THETAP=2*DATAN(T/2)
C   T2=T*T
C
C   AA0=4 + 2*AAA*T + T2
C   AA1=2*T2 - 8
C   AA2=4 - 2*AAA*T + T2
C   BB0=4 + 2*BBB*T + T2
C   BB1=AA1
C   BB2=4 - 2*BBB*T + T2
C
C   ALPHA=DSIN((THETAP-THETAC)/2)/
C   *      DSIN((THETAP+THETAC)/2)
C   ALPHA2=ALPHA*ALPHA
C
C   B0=AA0 - ALPHA*AA1 + ALPHA2*AA2

```



```

D0=BB0 - ALPHA*BB1 + ALPHA2*BB2
C
A0=T2*( 1 - 2*ALPHA + ALPHA2 )
A1=T2*( 2*ALPHA2 - 4*ALPHA + 2 )
A2=T2*( 1 - 2*ALPHA + ALPHA2 )
B1=(-2*ALPHA*AA0 + (ALPHA2+1)*AA1 - 2*ALPHA*AA2)/B0
B2=( ALPHA2*AA0 - ALPHA*AA1 +AA2 )/B0
C
C0=A0/D0
C1=A1/D0
C2=A2/D0
A1=A1/B0
A0=A0/B0
A2=A2/B0
D1=(-2*ALPHA*BB0 + (ALPHA2+1)*BB1 - 2*ALPHA*BB2)/D0
D2=( ALPHA2*BB0 - ALPHA*BB1 +BB2 )/D0
C
CHECK FOR UNITY D.C. GAIN OF H1(Z) AND H2(Z)
B0=(A0+A1+A2)/(1.+B1+B2)
D0=(C0+C1+C2)/(1.+D1+D2)
WRITE(6,14) A0,A1,A2,B1,B2,B0
14  FORMAT('0 A0= ',D13.6,'      A1= ',D13.6,'      A2= ',D13.6/
*      '      B1= ',D13.6,'      B2= ',D13.6/
*      '      D.C. GAIN = ',D13.6)
WRITE(6,15) C0,C1,C2,D1,D2,D0
15  FORMAT('0 C0= ',D13.6,'      C1= ',D13.6,'      C2= ',D13.6/
*      '      D1= ',D13.6,'      D2= ',D13.6/
*      '      D.C. GAIN = ',D13.6)
C
IA0=HFIX(SNGL(A0*2**15+.5))
IA1=HFIX(SNGL(A1*2**15+.5))
IA2=HFIX(SNGL(A2*2**15+.5))
IB1=HFIX(SNGL((-1.-B1)*32768+.5))
IB2=HFIX(SNGL(-B2*32768-.5))
IC0=HFIX(SNGL(C0*2**15+.5))
IC1=HFIX(SNGL(C1*2**15+.5))
IC2=HFIX(SNGL(C2*2**15+.5))
ID1=HFIX(SNGL((-1.-D1)*32768+.5))
ID2=HFIX(SNGL(-D2*32768-.5))
WRITE(6,16) IA0,IA1,IA2,IB1,IB2
WRITE(6,17) IC0,IC1,IC2,ID1,ID2
16  FORMAT('0 A0= ',Z4,'      A1= ',Z4,'      A2= ',Z4/
*      '      -1-B1= ',Z4,'      -B2= ',Z4)
17  FORMAT('0 C0= ',Z4,'      C1= ',Z4,'      C2= ',Z4/
*      '      -1-D1= ',Z4,'      -D2= ',Z4)
C
WRITE(6,18)
18  FORMAT('0 ZEROS AT :')
WRITE(7,21)
21  FORMAT('0'/'004004')
C
CALL QRTS8(A0,A1,A2,R1,R2)
WRITE(6,19) R1,R2
19  FORMAT(' ',F11.6,'      J',F11.6)

```



```

22      WRITE(7,22) R1,R2
      FORMAT(F12.7)
      CALL QRTS8(C0,C1,C2,R1,R2)
      WRITE(6,19) R1,R2
      WRITE(7,22) R1,R2
C
      WRITE(6,20)
20      FORMAT('0 POLES AT :')
      CALL QRTS8(1.D 00,B1,B2,R1,R2)
      WRITE(6,19) R1,R2
      WRITE(7,22) R1,R2
      CALL QRTS8(1.D 00,D1,D2,R1,R2)
      WRITE(6,19) R1,R2
      WRITE(7,22) R1,R2
      FMAX=10.**IFIX(SNGL(DLOG10(THETAF*10)+.9D 00))
      FMIN=10.**IFIX(SNGL(DLOG10(THETAF/10)-.9D 00))
      WRITE(7,22) FMIN,FMAX,T
      WRITE(7,21)
      WRITE(6,23)
23      FORMAT(' SIMULATION OF THE STEP RESPONCE')
      IF( S.EQ.0 ) STOP
      A0=IA0/2**15.
      A1=IA1/2**15.
      A2=IA2/2**15.
      B1=IB1/32768.
      B2=IB2/32768.
      C0=IC0/2**15.
      C1=IC1/2**15.
      C2=IC2/2**15.
      D1=ID1/32768.
      D2=ID2/32768.
C      CHECK FOR UNITY D.C. GAIN OF H1(Z) AND H2(Z)
      B0=(A0+A1+A2)/(-B1-B2)
      D0=(C0+C1+C2)/(-D1-D2)
      WRITE(6,14) A0,A1,A2,B1,B2,B0
      WRITE(6,15) C0,C1,C2,D1,D2,D0
      WRITE(6,18)
      WRITE(8,21)
C
      CALL QRTS8(A0,A1,A2,R1,R2)
      WRITE(6,19) R1,R2
      WRITE(8,22) R1,R2
      CALL QRTS8(C0,C1,C2,R1,R2)
      WRITE(6,19) R1,R2
      WRITE(8,22) R1,R2
C
      RB1=-(1+B1)
      RB2=-B2
      RD1=-(1+D1)
      RD2=-D2
      WRITE(6,20)
      CALL QRTS8(1.D 00,RB1,RB2,R1,R2)
      WRITE(6,19) R1,R2
      WRITE(8,22) R1,R2

```



```

CALL QRTS8(1.D 00, RD1, RD2, R1, R2)
WRITE(6, 19) R1, R2
WRITE(8, 22) R1, R2
FMAX=10.**IFIX(SNGL(DLOG10(THETAF*10)+.9D 00))
FMIN=10.**IFIX(SNGL(DLOG10(THETAF/10)-.9D 00))
WRITE(8, 22) FMIN, FMAX, T
WRITE(8, 21)
400 WRITE(6, 26)
26  FORMAT(' ', 8X, 'ENTER MAGNITUDE OF STEP IN HEX, (Z4)')
READ(5, 25) S
25  FORMAT(Z4)
IF( S.EQ.0 ) STOP
SS=S*2**8
RS=SS
O=0
P=0
Q=0
R=0
DO 300 I=1, 300
C
W=O
O=P
Y=Q
Q=R
P=O
R=O
C
CALL IMULT(SS, A0, W)
CALL IMULT(SS, A1, O)
CALL IMULT(SS, A2, P)
C
CALL IMULT(W, B1, O)
O=O+W
CALL IMULT(W, B2, P)
CALL IMULT(W, C0, Y)
CALL IMULT(W, C1, Q)
CALL IMULT(W, C2, R)
C
CALL IMULT(Y, D1, Q)
Q=Q+Y
CALL IMULT(Y, D2, R)
N(I)=I
IRESP(I)=Y/2**8
300 RESP(I)=Y/RS
24  WRITE(6, 24)(( IRESP(I), RESP(I)), I=1, 300)
FORMAT(' ', 2(Z8, 3X, F9.4, 5X))
GO TO 400
END
C
C
SUBROUTINE QRTS8(A, B, C, R1, R2)
C  DUBBLE PRECISION ROOTS OF A QUADRATIC
COMPLEX*16 R1, R2
REAL*8 A, B, C, R, A2

```



```

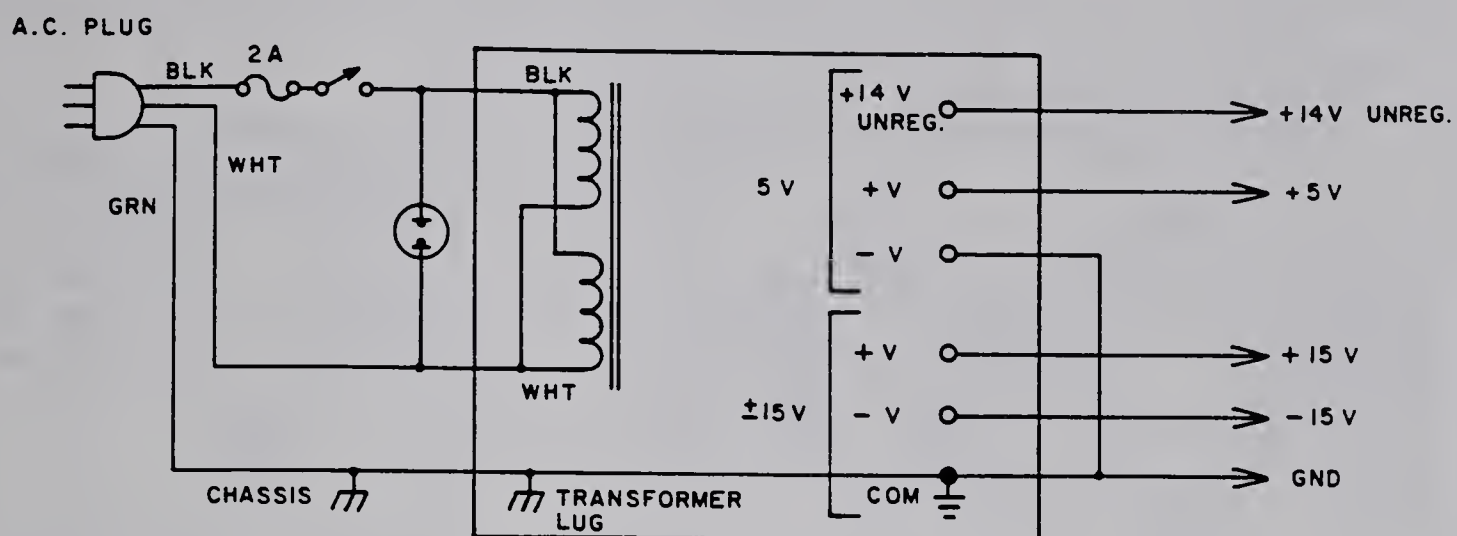
      R1=DCMPLX(0.0D 00,0.0D 00)
      R2=R1
      R=B*B-4*A*C
      IF( A .NE. 0.0D 00 ) GO TO 50
C     ONLY ONE ROOT
      R1=DCMPLX(-C/B,0.0D 00)
      GO TO 200
50    A2=2.0D 00*A
      IF( R .LT. 0.0D 00 ) GO TO 100
C     ROOTS ARE REAL
      R1=DCMPLX((-B+DSQRT(R))/A2,0.0D 00)
      R2=DCMPLX((-B-DSQRT(R))/A2,0.0D 00)
      GO TO 200
C     ROOTS ARE COMPLEX
100   R1=DCMPLX(-B/A2,DSQRT(-R)/A2)
      R2=DCMPLX(-B/A2,-DSQRT(-R)/A2)
200   RETURN
      END
C
C
      SUBROUTINE IMULT(II,M,IO)
      INTEGER  II,IO
      REAL*8  M,R
      R=M*II
C     IF(R) 10,20,30
10    IO=IO+R-0.0
20    RETURN
30    IO=IO+R+.5
      RETURN
      END

```

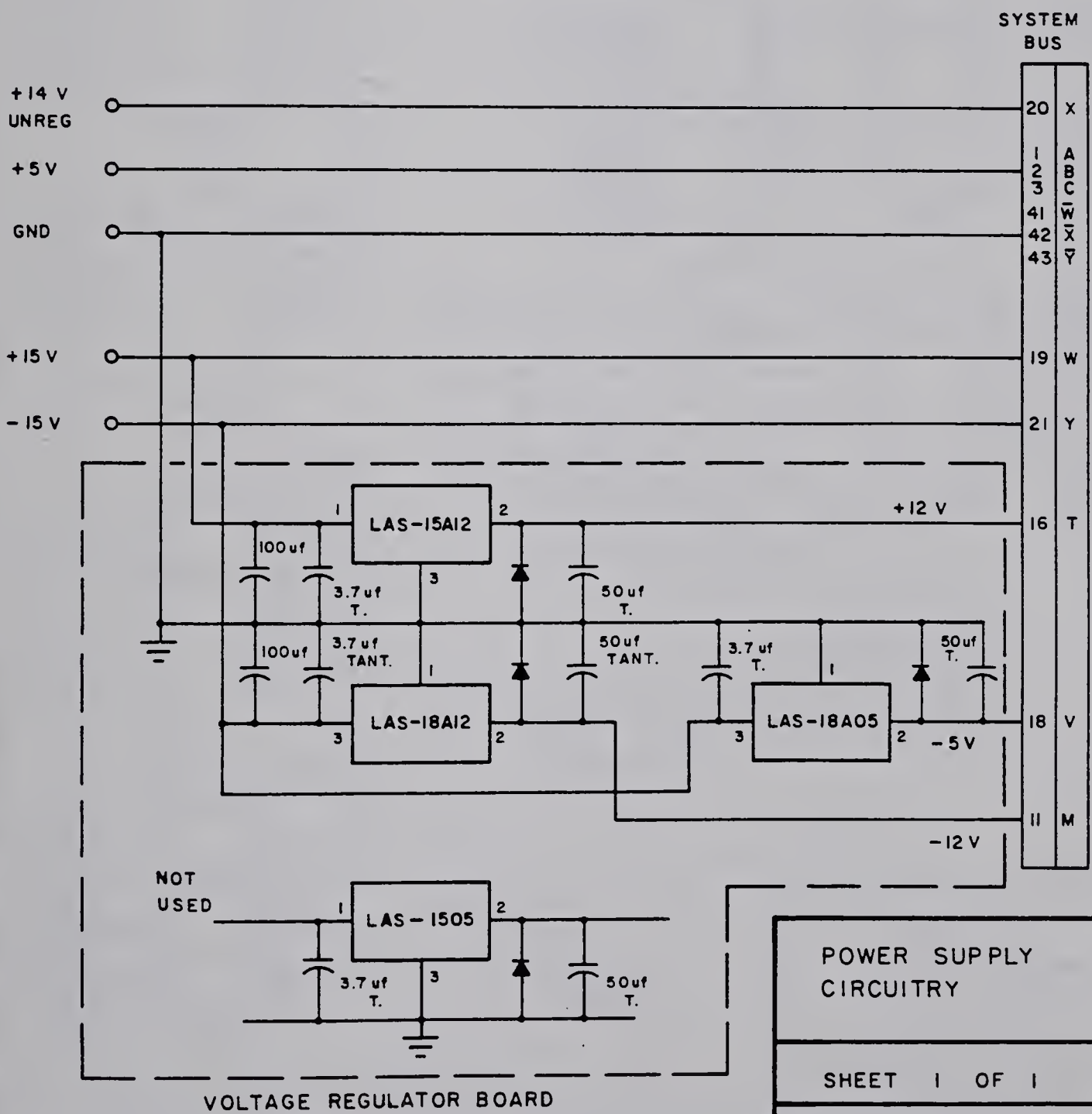

Appendix D -- Schematic Diagrams

The schematic diagram for Board 1 of the microcomputer system can be found in the manual, **M68MM01A/1A2 Monoboard Microcomputer Micromodule 1A/1A2**, (by Motorola Incorporated, Phoenix, May 1979). The only alteration on this board is that a jumper was installed to set the ACIA for a baud rate of 9400. (See manual.)





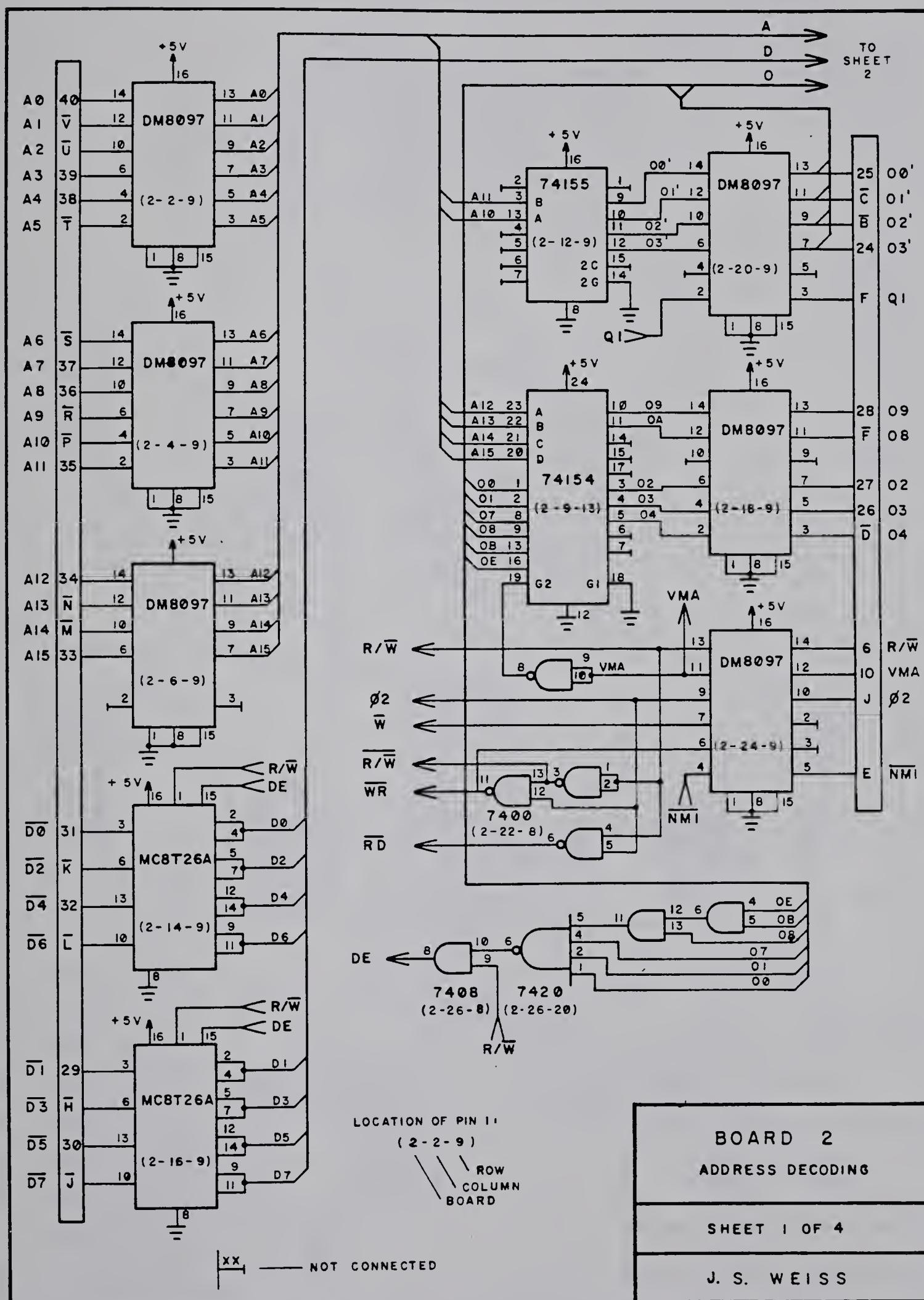
LAMBDA REGULATED POWER SUPPLY
LOT -V-5152-A

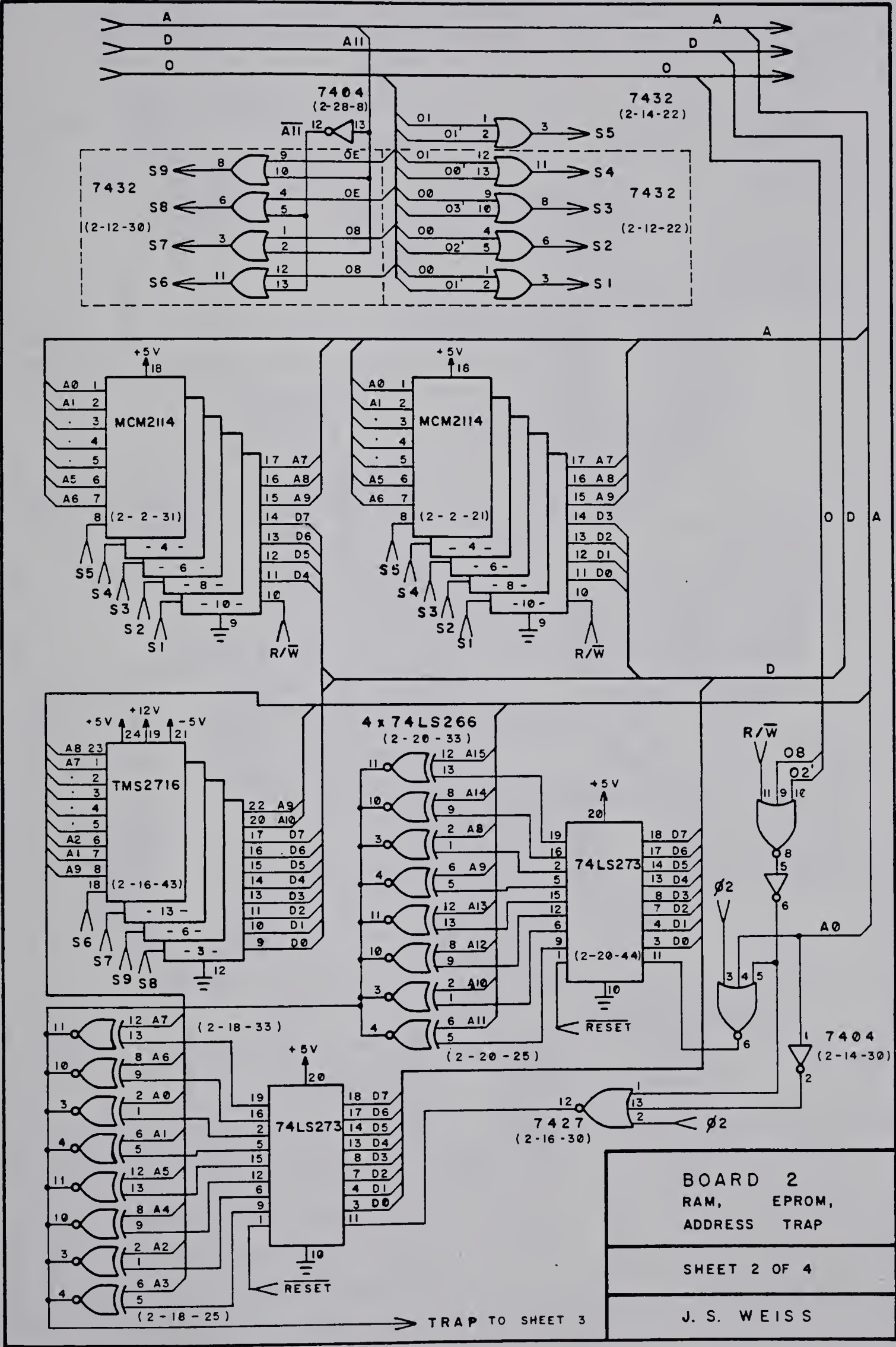


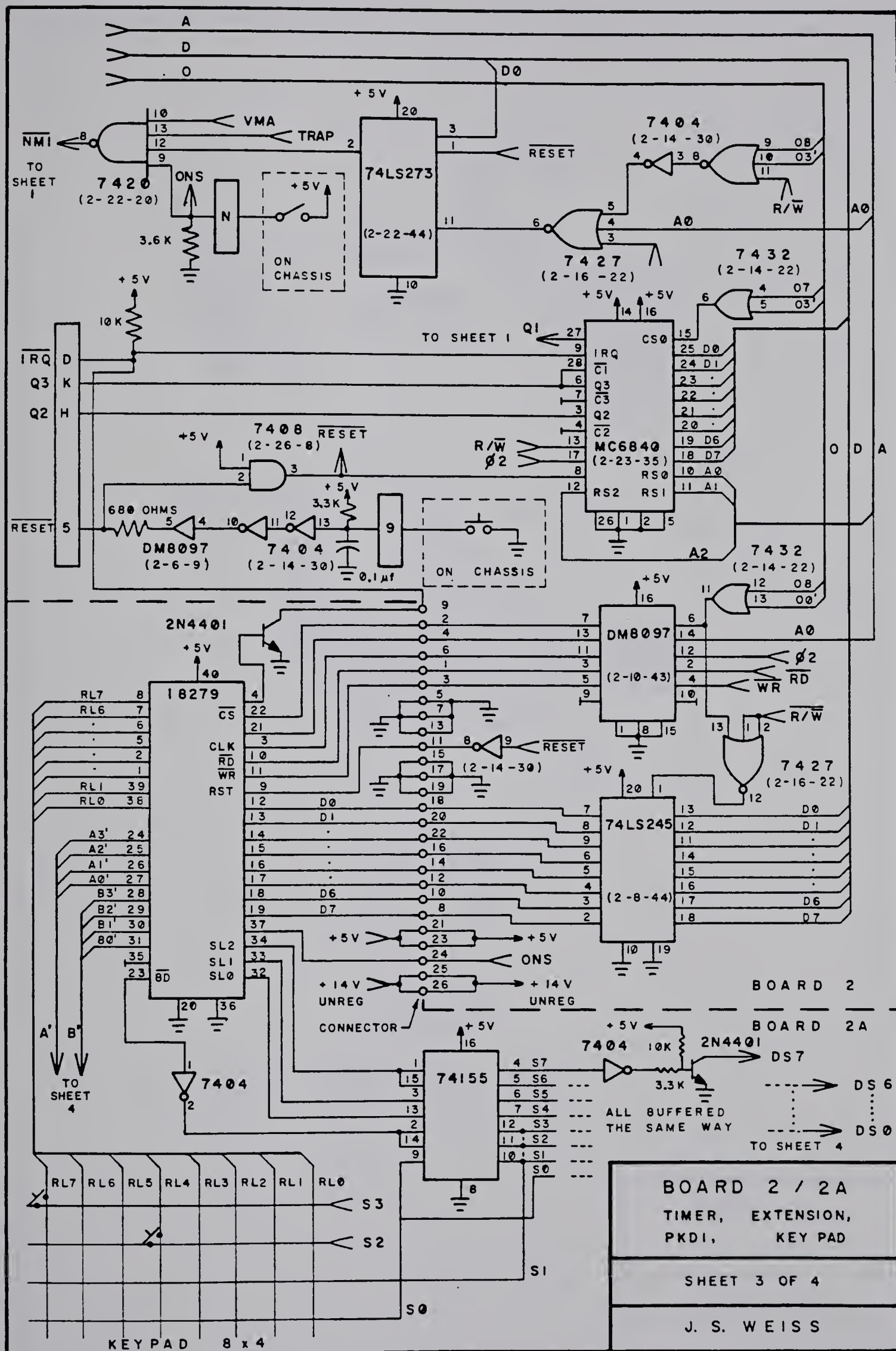
POWER SUPPLY
CIRCUITRY

SHEET 1 OF 1

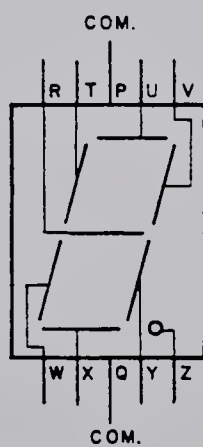
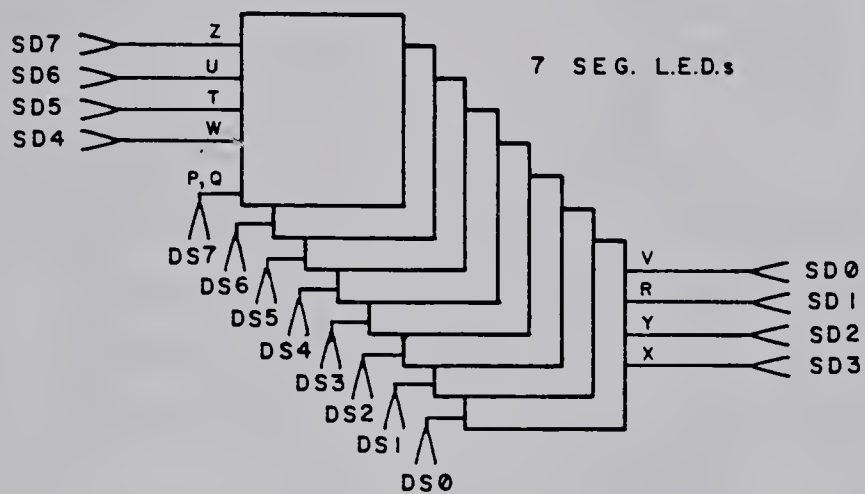
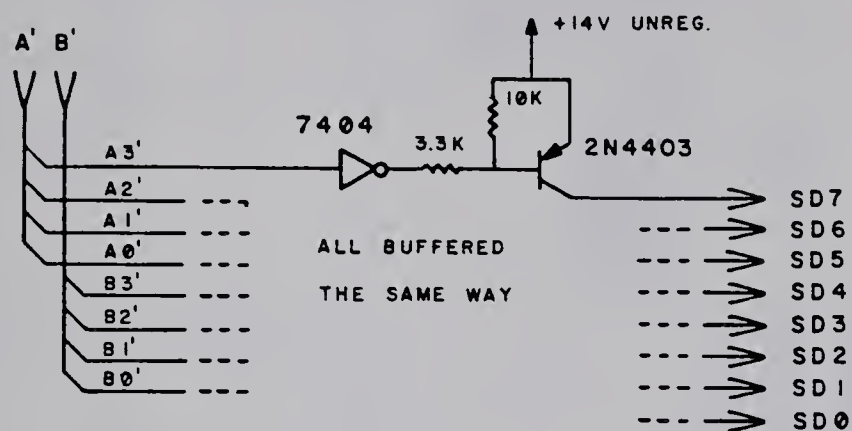
J. S. WEISS











PINOOTS OF THE 7 SEGMENT
COMMON CATHODE L.E.D.s

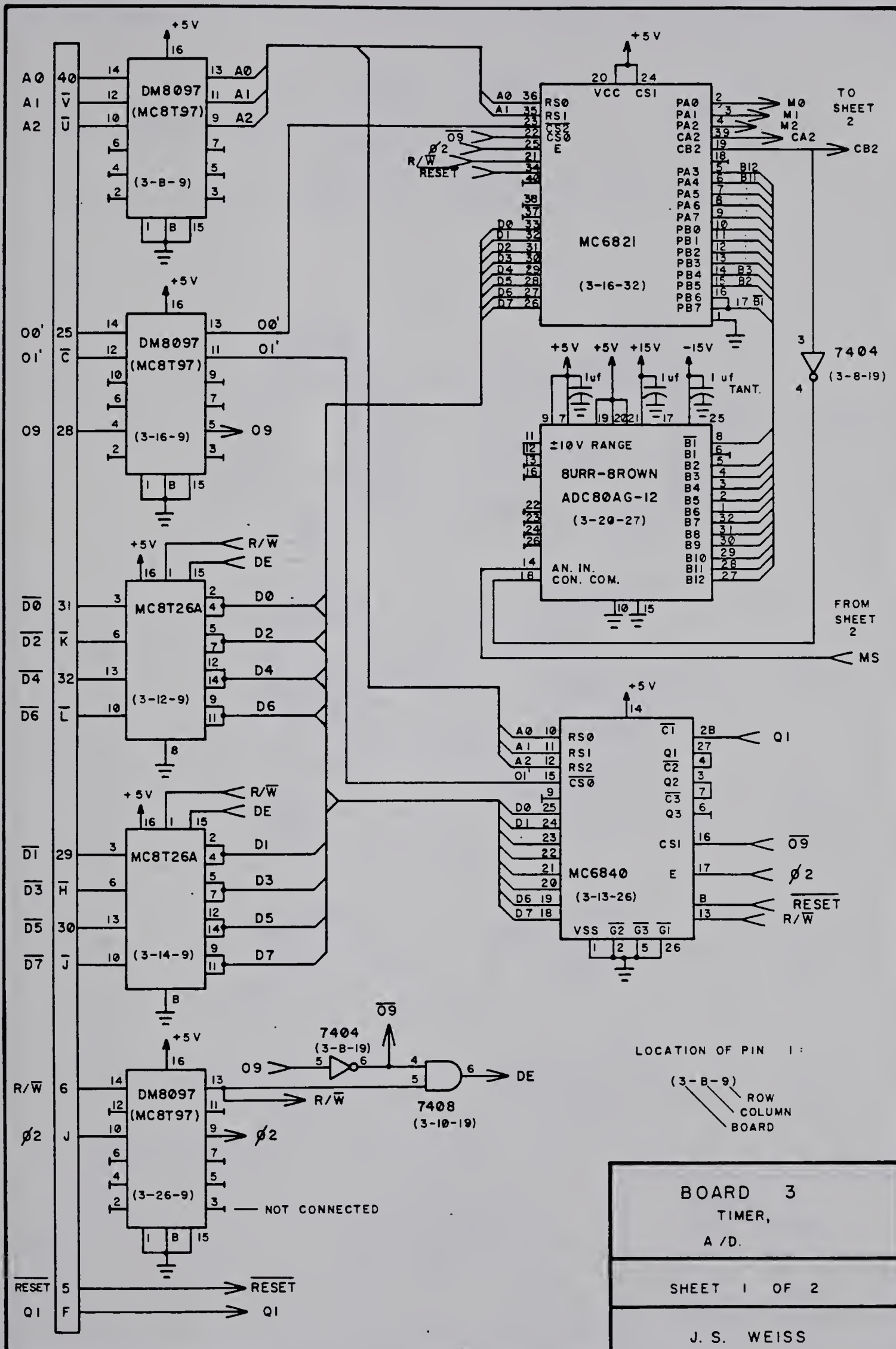
BOARD 2A

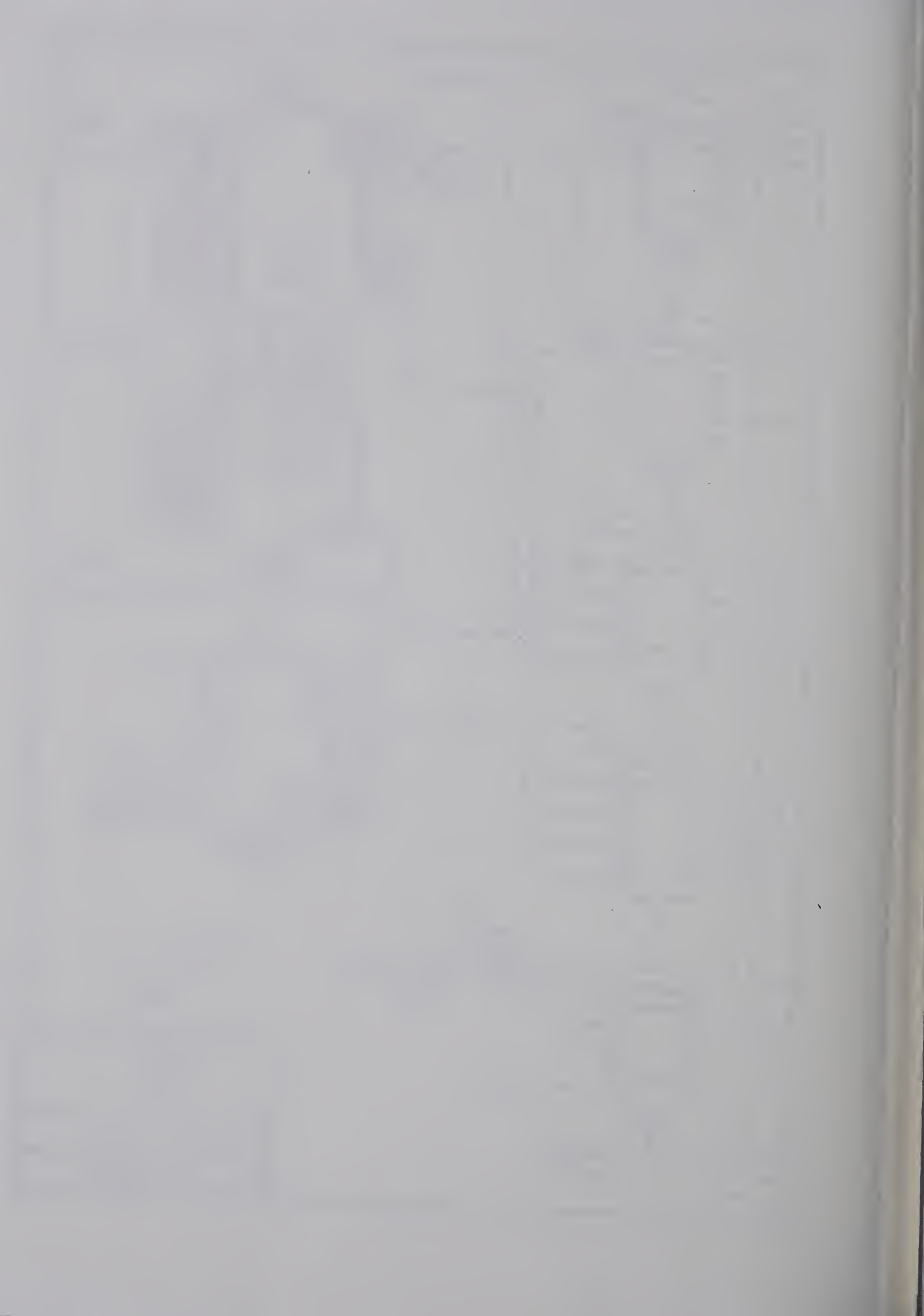
L.E.D. DISPLAY

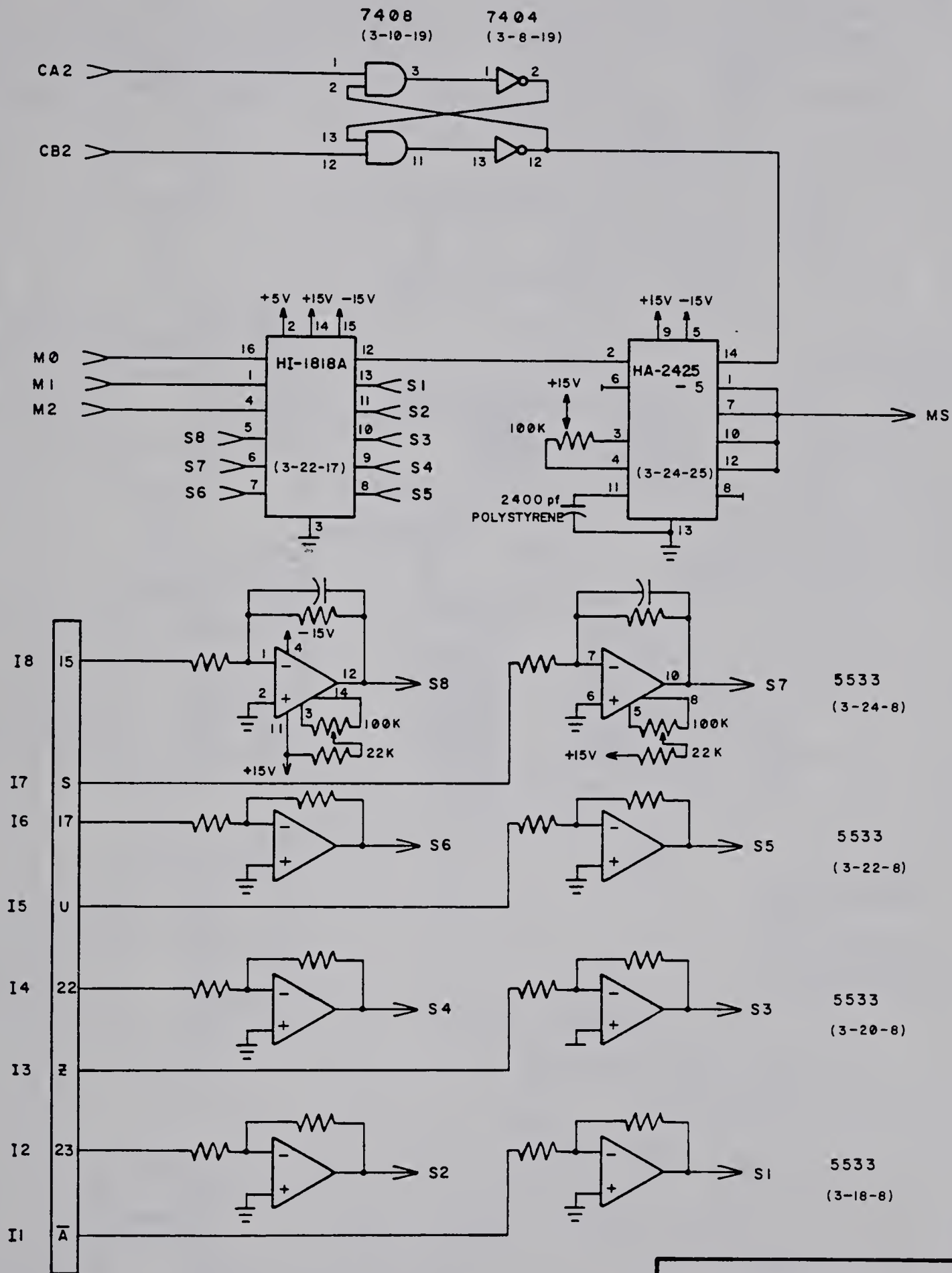
SHEET 4 OF 4

J. S. WEISS







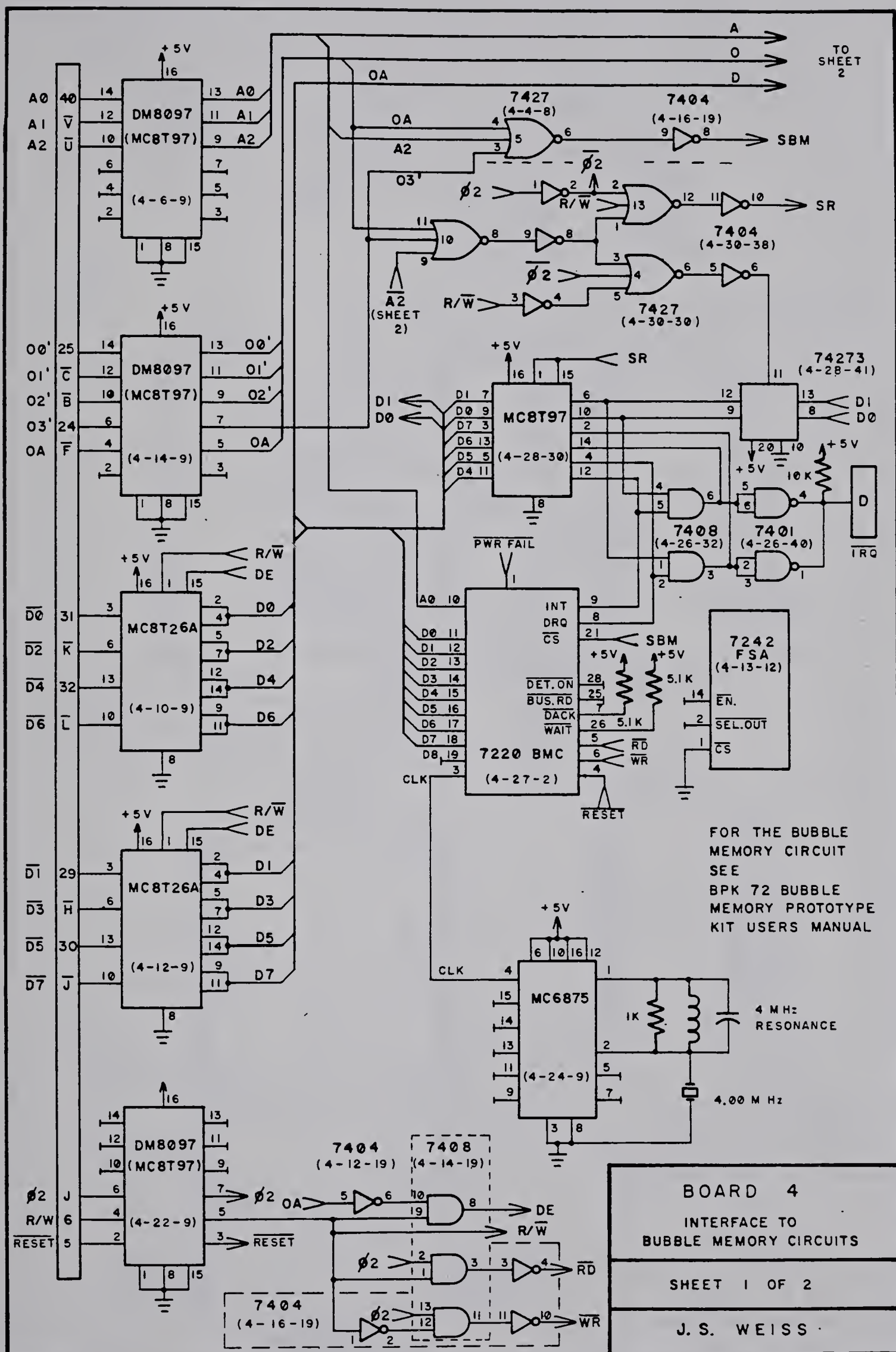


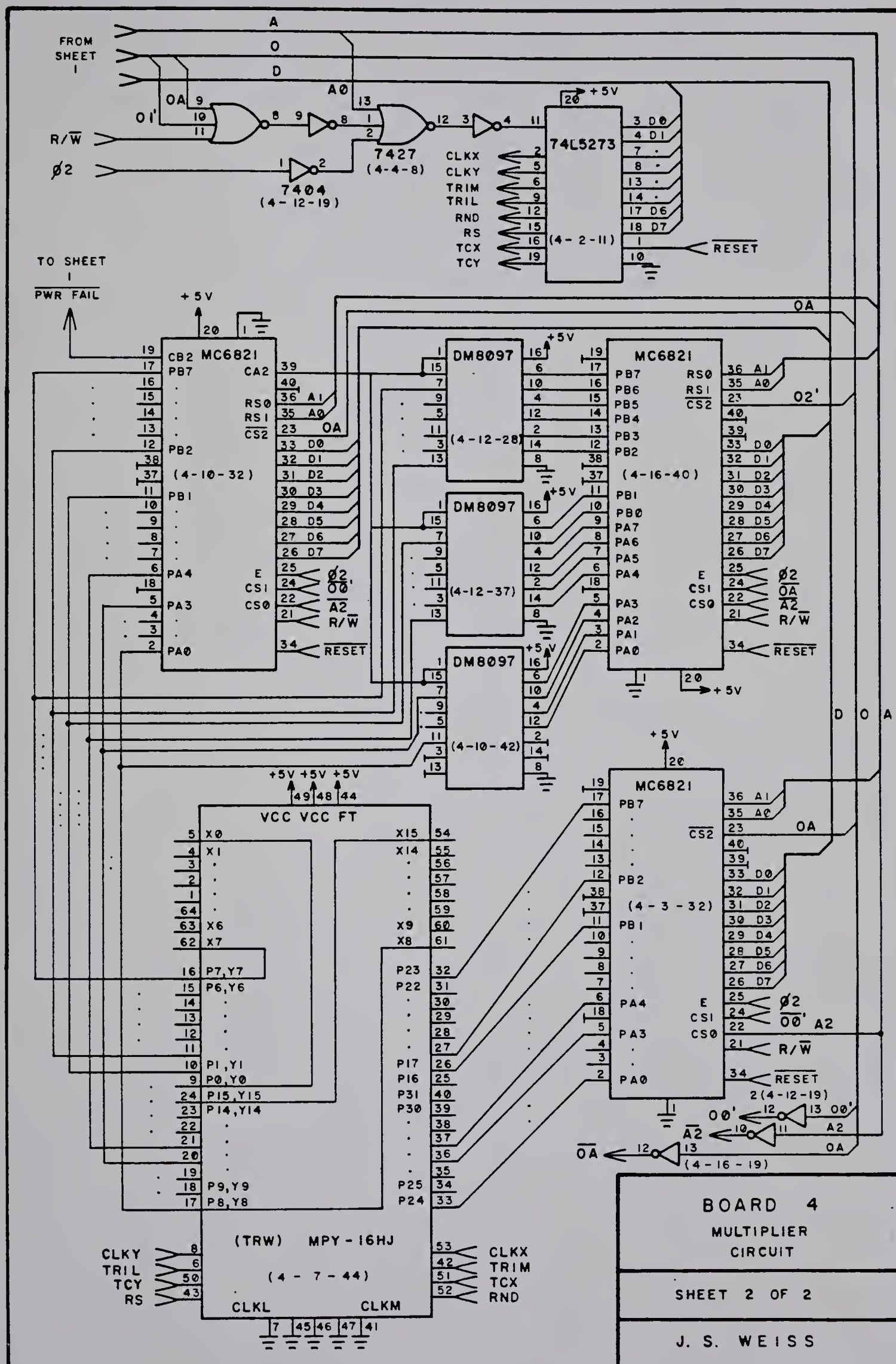
BOARD 3
INPUT BUFFERS,
S/H, MUX,

SHEET 2 OF 2

J. S. WEISS







B30354